SIMULIA

3DEXPERIENCE

ABAQUS 2026 FD04
ANALYSIS GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....7
Abaqus Analysis Guide....8
What's New....9
Analysis Procedures....82
Introduction....83
Solving Analysis Problems....84
Defining an Analysis....87
General and Perturbation Procedures....98
Multiple Linear Load Case Analysis....104
Multiple Nonlinear Load Case Analysis....110
Step Cycling....116
Direct Linear Equation Solver....119
Iterative Linear Equation Solver....121
Static Stress/Displacement Analysis....126
About Static Stress Analysis Procedures....127
Static Stress Analysis....128
Quasi-Static Analysis....134
Direct Cyclic Analysis....138
Eigenvalue Buckling Prediction....150
Unstable Collapse and Postbuckling Analysis....160
Linear Elastic Fatigue Crack Growth Analysis....166
Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach....180
Dynamic Stress/Displacement Analysis....194
About Dynamic Analysis Procedures....195
Implicit Dynamic Analysis Using Direct Integration....221
Explicit Dynamic Analysis....234
Direct-Solution Steady-State Dynamic Analysis....251
Natural Frequency Extraction....262
Complex Eigenvalue Extraction....282
Transient Modal Dynamic Analysis....290
Mode-Based Steady-State Dynamic Analysis....301
Subspace-Based Steady-State Dynamic Analysis....320
Response Spectrum Analysis....335
Random Response Analysis....351
Steady-State Transport Analysis....367
Steady-State Transport Analysis 367
Defining the Model....368
Defining Reference Frame Motions....370
Contact Conditions....371
Incrementation....373
Convergence in a Steady-State Transport Analysis....374
Initial Conditions....377
Boundary Conditions....377

Loads....377
Predefined Fields....378
Material Options....379
Elements....380
Output....380
Limitations....381
Input File Template....381
Heat Transfer and Thermal-Stress Analysis....383
About Heat Transfer Analysis Procedures....384
Uncoupled Heat Transfer Analysis....386
Fully Coupled Thermal-Stress Analysis....397
Adiabatic Analysis....410
Electromagnetic Analysis....416
Electromagnetic Analysis Procedures....417
Piezoelectric Analysis....419
Coupled Thermal-Electrical Analysis....424
Fully Coupled Thermal-Electrical-Structural Analysis....435
Eddy Current Analysis....444
Magnetostatic Analysis....456
Electrochemical Analysis....461
About Electrochemical Analysis Procedures....462
Coupled Thermal-Electrochemical Analysis....464
Fully Coupled Thermal-Electrochemical-Structural Analysis....485
Fully Coupled Thermal-Electrochemical-Structural–Pore Pressure Analysis....491
Modeling Solid Electrodes in Lithium Metal Batteries....499
Modeling Aging in Batteries....504
Modeling Thermal Runaway in Batteries....513
Modeling Solid Electrolytes and Solid-State Batteries....521
Parameter Table Type Reference....536
Property Table Type Reference....578
Coupled Pore Fluid Flow and Stress Analysis....611
Coupled Pore Fluid Diffusion and Stress Analysis....612
Undrained Pore Fluid Flow and Stress Analysis....627
Geostatic Stress State....634
Mass Diffusion Analysis....643
Governing Equations....643
Fick's Law....644
Units....645
Steady-State Analysis....645
Transient Analysis....645
Initial Conditions....647
Boundary Conditions....647
Loads....647
Predefined Fields....649
Material Options....649
Elements....649
Output....650
Input File Template....651

References....652
Coupled Acoustic-Structural Analysis....653
Procedures Available for Acoustic Analysis....654
Defining Translational or Rotational Underlying Flow Velocity in Abaqus/Standard....658
Updating the Acoustic Domain during a Large-Displacement Abaqus/Standard Analysis....659
Initial Conditions....659
Boundary Conditions....660
Loads....664
Predefined Fields....665
Material Options....665
Elements....665
Output....669
Input File Template....670
Abaqus/Aqua Analysis....675
Procedures Available for Aqua Analysis....675
Defining an Abaqus/Aqua Problem....676
Defining the Fluid Properties....676
Defining a Steady Current....676
Defining Gravity Waves....677
Defining a Wind Velocity Profile....685
Initial Conditions....686
Boundary Conditions....686
Loads....686
Predefined Fields....698
Material Options....698
Elements....698
Output....698
Input File Template....698
Annealing....700
The Annealing Process....700
Initial Conditions....701
Boundary Conditions....701
Loads....701
Predefined Fields....701
Material Options....701
Elements....701
Output....701
Input File Template....701
One-Step Inverse Analysis....703
Introduction....703
Part Definition....704
Incrementation....704
Controlling the Solution Accuracy....704
Initial Conditions....705
Boundary Conditions....705
Loads....706
Predefined Fields....706
Material Options....706

Elements....706
Constraints....706
Output....706
Limitations....707
Input File Template....707
References....707
Analysis Solution and Control....708
Solving Nonlinear Problems....709
The Solution of Nonlinear Problems....709
Convergence....710
Solution Method....712
Automatic Incrementation Control....718
Automatic Stabilization of Unstable Problems....719
Input File Template....724
Analysis Convergence Controls....725
About Convergence and Time Integration Criteria....726
Commonly Used Control Parameters....727
Convergence Criteria for Nonlinear Problems....734
Time Integration Accuracy in Transient Problems....745
Modifying the Default Convergence Checks....748
Analysis Techniques....752
About Analysis Techniques....753
Analysis Continuation Techniques....753
Modeling Abstractions....753
Special-Purpose Techniques....754
Adaptivity Techniques....755
Optimization Techniques....755
Eulerian Analysis....755
Particle Methods....755
Sequentially Coupled Multiphysics Analyses....755
Co-Simulation....755
Extending Abaqus Analysis Functionality....755
Design Sensitivity Analysis....756
Parametric Studies....756
Availability of Analysis Techniques....756
Analysis Continuation Techniques....758
Restarting an Analysis....759
Importing and Transferring Results....773
Modeling Abstractions....828
Substructures....829
Submodeling....880
Matrices....930
Symmetric Model Generation, Results Transfer, and Analysis of Cyclic Symmetry Models....956
Periodic Media Analysis....983
Meshed Beam Cross-Sections....989
Using the Extended Finite Element Method....999
Reduced Basis Generation....1057
Special-Purpose Techniques....1062

Inertia Relief....1063
Element and Contact Pair Removal and Reactivation....1074
Progressive Element Activation....1083
Introducing a Geometric Imperfection into a Model....1088
Fracture Mechanics....1092
Surface-Based Fluid Modeling....1146
Mass Scaling....1184
Selective Subcycling....1198
Steady-State Detection....1200
CZone Analysis....1206
Additive Manufacturing Process Simulation....1231
About Additive Manufacturing Process Simulation....1232
Toolpath-Mesh Intersection Module....1235
Thermomechanical Simulation of Additive Manufacturing Processes....1246
Eigenstrain-Based Simulation of Additive Manufacturing Processes....1252
Special-Purpose Techniques for Additive Manufacturing....1258
Adaptivity Techniques....1351
About Adaptivity Techniques....1352
ALE Adaptive Meshing....1357
Adaptive Remeshing....1426
Mesh-to-Mesh Solution Mapping....1448
Fatigue Crack Propagation with Adaptive Remeshing....1453
Optimization Techniques....1472
About Structural Optimization....1473
Optimization Models....1485
Eulerian Analysis Techniques....1514
Eulerian Analysis....1515
Defining Eulerian Boundaries....1525
Eulerian Mesh Motion....1528
Defining Adaptive Mesh Refinement in the Eulerian Domain....1533
Particle Methods....1537
Discrete Element Method....1538
Continuum Particle Analyses....1554
Particle Generator....1577
Lumped Kinetic Molecular Method....1588
Sequentially Coupled Multiphysics Analyses....1596
General Capability for Importing External Fields....1597
Predefined Fields for Sequential Coupling....1607
Sequentially Coupled Thermal-Stress Analysis....1612
Sequentially Coupled Injection Molding to Stress Analysis....1615
Predefined Loads for Sequential Coupling....1620
Co-Simulation....1622
About Co-Simulation....1623
Preparing an Abaqus Analysis for Co-Simulation....1629
Co-Simulation between Abaqus Solvers....1644
System-Level Modeling between Logical and Physical Interactions....1661
Field Mapper....1664
Field Mapping: General Remarks....1664

Domain Coupling for Multiphysics and Multiscale Simulations....1665
Mapping Operations....1665
Mapping Algorithms....1666
Modifying the Search Position Tolerance Distance....1667
Selecting the Base Configuration....1668
Unmapped Members Treatment....1668
Specifying Tensor Averaging....1669
Design Sensitivity Analysis....1670
Design Sensitivity Analysis Overview....1671
Direct Design Sensitivity Analysis....1673
Adjoint Design Sensitivity Analysis....1684
Parametric Studies....1708
Scripting Parametric Studies....1709
Parametric Studies: Commands....1727

Trademarks and Legal Notices

Trademarks

Abaqus, **3D**EXPERIENCE®, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

Legal Notices

Abaqus and this documentation may be used or reproduced only in accordance with the terms of the software license agreement signed by the customer, or, absent such agreement, the then current software license agreement to which the documentation relates.

This documentation and the software described in this documentation are subject to change without prior notice.

Dassault Systèmes or its Affiliates shall not be responsible for the consequences of any errors or omissions that may appear in this documentation.

© Dassault Systèmes Americas Corp., 2026.

For a full list of the third-party software contained in this release, please go to the Legal Notices in the Abaqus 2026 HTML documentation, which can be obtained from a documentation installation, or in the SIMULIA Established Products 2026 Program Directory, which is available on *www.3ds.com*.

Abaqus Analysis Guide

The Abaqus Analysis Guide describes the analysis types (static stress analysis, dynamics, eigenvalue extraction, etc.) that are available as well as various analysis techniques available in Abaqus such as submodeling, removing elements or surfaces, and importing results from a previous simulation to define the initial conditions for the current model.

This guide is a part of the Abaqus® documentation collection, which describes all the capabilities of the Abaqus finite element analysis technology used in SIMULIA® applications.

Detailed discussions of the differences between how Abaqus/Standard and Abaqus/Explicit solve finite element analyses are provided.

What's New

This page describes recent changes in Abaqus Analysis.

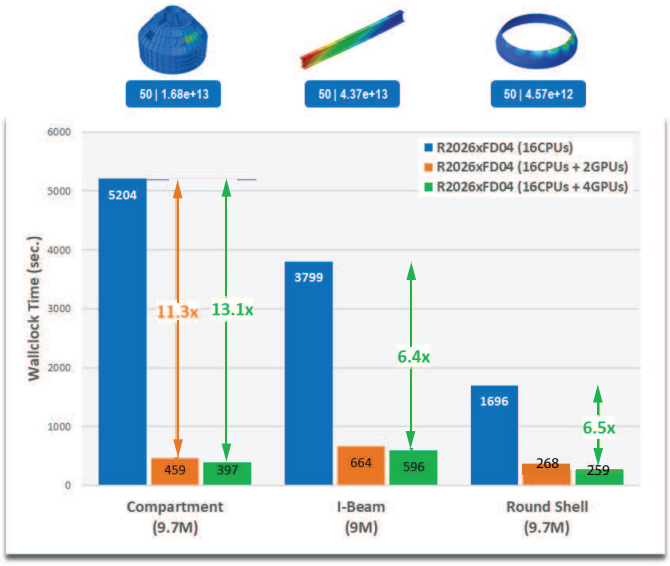
2026 FD04

GPU-Based Subspace Iteration Eigensolver

Performance of the subspace iteration eigensolver is improved significantly on systems with supported GPGPUs.

Benefits: You can reduce the run time of linear buckling analyses by leveraging GPGPU acceleration. When running on a single compute system using fewer than 32 CPU cores, one or more GPGPU cards can be used to accelerate the subspace iteration eigensolver for substantial speedups.

For example, the figure below highlights the performance gains achieved by the new subspace-based iteration solver across three representative buckling benchmark models, where each model has over 9M DOFs requiring over 1×10^{13} floating-point operations to factorize the global stiffness matrix. The performance data was obtained on a machine with Intel Xeon 6442Y processors (Sapphire Rapids, 2.6 GHz, 48 cores), 1700 GB RAM, and 4x NVIDIA H100 (96 GB/GPU).



The blue bars show the run time of Abaqus/Standard using the subspace iteration solver in shared memory parallel mode with 16 CPU cores only, whereas the orange and green bars correspond to the GPU-based subspace iteration solver with 2 GPUs and 4 GPUs, respectively. The results demonstrate that the GPU-based subspace iteration solver shows exceptional scaling up to 4 GPUs—with 13x speedup in the aerospace compartment model. Moreover, the GPU speedup increases as the number of subspace iterations increases, demonstrating that the new subspace iteration solver is especially efficient for problems requiring many iterations to converge.

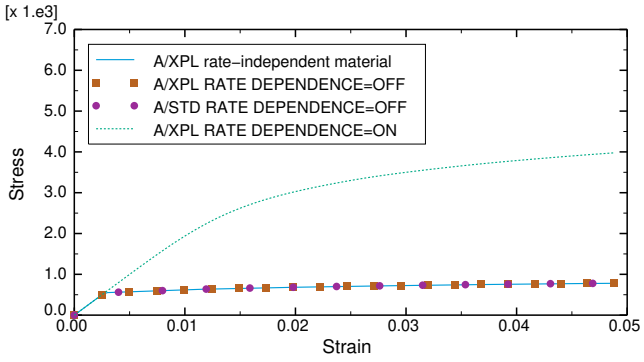
For more information, see *Parallel Execution in Abaqus/Standard*.

Controlling Material Rate Dependence in Explicit Dynamic Simulations

You can now control whether or not to consider material rate–dependence in an explicit dynamic step or coupled dynamic thermal-stress step.

Benefits: You can more easily run explicit dynamic simulations with rate-independent or long-term material response.
In earlier releases, you needed to remove rate-dependent properties from the material definitions to ignore material rate-dependence in explicit dynamic steps. Now, you can set the RATE DEPENDENCE parameter on the *DYNAMIC or *COUPLED TEMPERATURE DISPLACEMENT options equal to ON or OFF to include or ignore these effects. This can be helpful for explicit dynamic procedures simulating long-duration quasi-static processes or for applying long-term preloads preceding sudden dynamic events.

The figure below illustrates the agreement of the simulations ignoring material rate-dependent behaviors with reference simulations where the corresponding material definitions do not contain rate-dependent properties.



For more information, see *Explicit Dynamic Analysis* and *Fully Coupled Thermal-Stress Analysis*.

Supporting Multiple Linear Load Cases in a Submodel Analysis

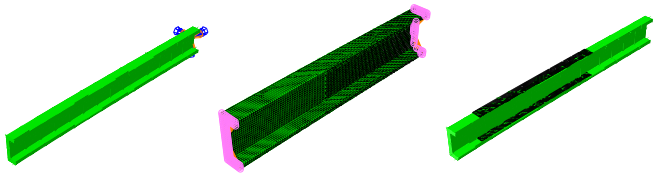
A step within a submodel can now refer to a step of a global model containing multiple linear load cases.

Benefits: You no longer need to create a separate perturbation step for each linear load case in the global model and the submodel, which reduces memory and execution time.

A multiple linear load case analysis is equivalent conceptually to a multiple-step analysis in which the load case definitions are mapped to consecutive perturbation steps. However, a multiple linear load case analysis might be more efficient than an equivalent multiple-step analysis.

Submodeling is used to study a local region in detail using results from a previously run larger model to drive conditions at submodel cuts. When a global model analysis contains multiple linear load cases, you can now use equivalent multiple linear load cases in the submodel analysis to study the behavior in the submodel region.

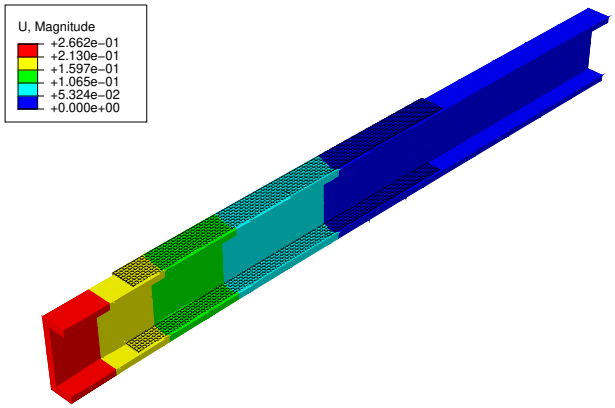
The figure below shows a global model in which a cantilever beam is modeled with beam elements (left), a submodel meshed with solid elements (center), and an overlay of the global model and submodel (right).



The general submodeling capability is used such that displacement and rotation results from the global model drive displacements and rotations via internally generated distributed couplings. The global and submodel simulations each have a three-step sequence as follows:

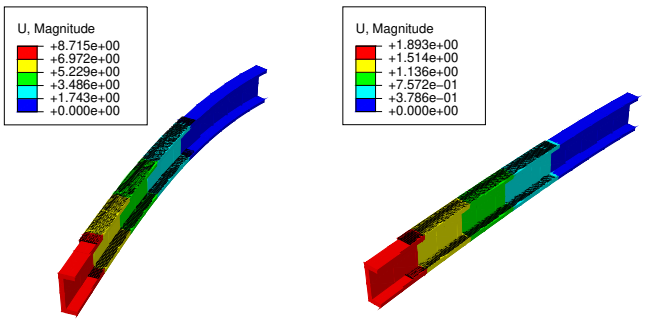
1. A linear perturbation step with two transverse load cases.
2. A general step with tensile loading and geometric nonlinearity considered.
3. Another linear perturbation step with the same two transverse load cases as in Step 1.

Step 2 is unaffected by Step 1 because Step 1 is a perturbation Step, but it causes Step 1 and Step 3 to have different base states. Step 2 applies axial tensile loading in a general step with nonlinear geometry, which causes stretching as shown in the figure below. Here, the displacement results from the global model and submodel are overlaid and in good agreement.

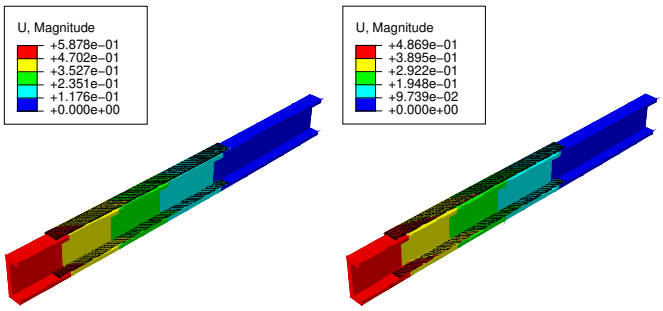


Step 1 involves a linear load case with transverse loading in the flange direction and another linear load case in the web direction. Step 3 has loading identical to that in Step 1, but it has a different base state than Step 1 due to inclusion of the load from nonlinear Step 2.

Below we compare perturbation solutions for the first and second load cases of Step 1 and Step 3, with results from the global model and submodel overlaid. The following figure shows displacement plots for the first load case of perturbation Step 1 (left) and Step 3 (right), causing bending of the beam:



The following figure shows displacement plots for the second load case of perturbation Step 1 (left) and Step 2 (right), causing bending of the beam.



The results for Step 1 and Step 3 differ significantly, even though the perturbation loads match, due to different base states. The base state for Step 1 is stress free. The base state for Step 3 includes the axial tensile stress resulting from Step 2, which leads to stress stiffening that reduces the amount of bending compared to the stress-free base state.

For more information, see *Using Multiple Linear Load Cases in a Submodel*.

Time Synchronization Between the Global Model and Submodel

You can now control the default behavior of time synchronization between the submodel analysis and the global model analysis.

Benefits: You can more easily understand the submodel setup.

By default, Abaqus now does the following:

- Associates the *i*th general step of a submodel with the *i*th general step of the global model for general submodeling.

  Abaqus issues an error message if the submodel has more general steps than the global model.

- Associates the *j*th perturbation step after the *i*th general step with the *j*th perturbation step after the *i*th general step of the global model.

  Abaqus issues an error message if the submodel has more perturbations steps than the global model after the corresponding general steps.

For more information, see *Referring to the Step and Increment of the Global Model*.

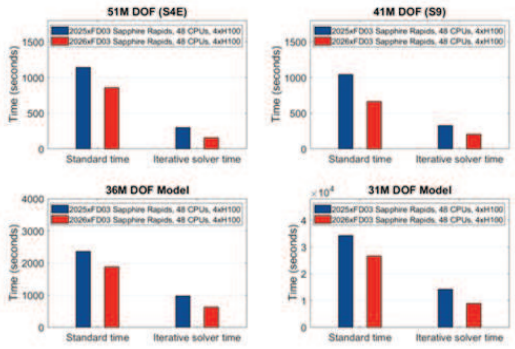
2026 FD03

GPU AMG Iterative Solver

Performance is improved significantly for the multi-GPU iterative solver, depending on model details and hardware configuration.

Benefits: Analysis run time is reduced.

Benchmark testing was conducted across four representative models to compare the performance of the GPU iterative solver in 2025 FD03 and 2026 FD03. All simulations were executed on a system equipped with 48-core Sapphire Rapids processors and four H100 GPUs. The results demonstrate a substantial reduction in total analysis time with the new implementation.



AMS GPU Acceleration for Frequency Extraction

AMS eigensolver performance for frequency extraction analysis is improved on systems equipped with high CPU core counts and NVIDIA GPUs with more than 80GB of memory.

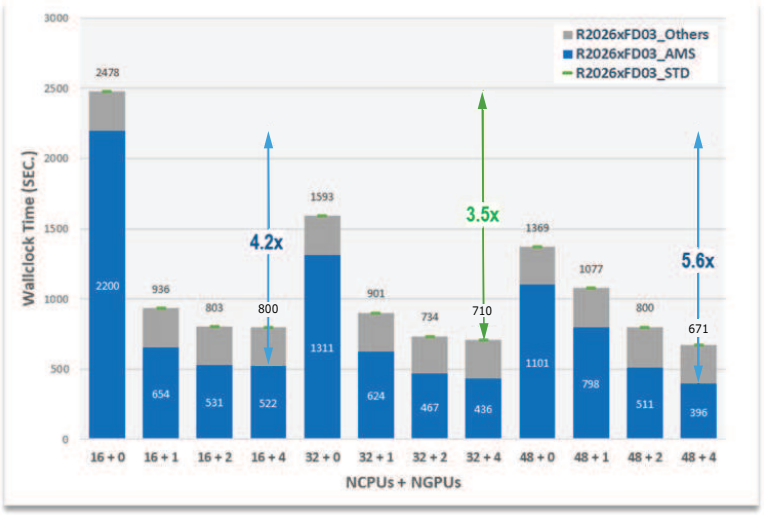
Benefits: The AMS eigensolver delivers significant speedups for frequency extraction analysis for large-scale bulky models and vehicle body models requiring over 20,000 eigenmodes.

With the advent of the high-performance NVIDIA GPUs featuring more than 80 GB memory, the GPU acceleration strategy for the AMS eigensolver has been redesigned. All three AMS phases now more effectively utilize large-memory GPUs, significantly improving overall GPU acceleration performance.

For the AMS reduction phase, we optimize the dynamic scheduling strategy to better saturate GPUs with larger workloads. In earlier releases, GPU acceleration performance often peaked at 16 CPUs when using multiple GPUs due to scheduling overhead costs. Now, an enhanced dynamic scheduling strategy efficiently distributes

more computational tasks to the large-memory GPUs, particularly on systems with higher CPU core counts (> 32 CPUs).

As an example, the figure below illustrates the GPU acceleration performance in 2026 FD03 for a 7.6M DOF stator model requiring 2.3e+15 floating-point operations to factorize four system operators (stiffness, mass, viscous, and structural damping matrices). The performance data was obtained on a machine equipped with Intel Xeon 6442Y processors (Sapphire Rapids, 2.6 GHz, 48 cores), 1700 GB RAM, and 4x NVIDIA H100 (96 GB) GPUs.

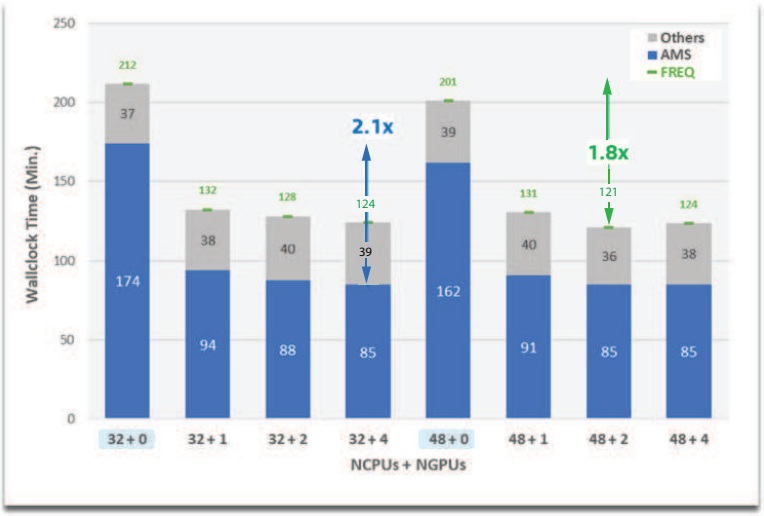


The blue columns represent the AMS run time, the gray columns represent the run time for the other phases (excluding the AMS run time from the total Abaqus/Standard time), and the small green dashes represent the total Abaqus/Standard run time. Abaqus/Standard using the AMS eigensolver demonstrates strong GPU-acceleration scaling up to 48 CPUs. Peak performance for this model is achieved using a configuration of 48 CPUs and 4 GPUs.

In addition, the GPU acceleration performance of the AMS eigensolver is further improved for a vehicle body model extracting more than 20,000 eigenmodes; this workload was previously constrained by GPUs with less than 80 GB of memory.

A new GPU dense eigensolver within the AMS reduced eigensolution phase can process large dense eigenproblems with significantly lower memory requirements. In addition, a new GPU acceleration for modal matrix projection including structural and viscous damping matrices and right-hand-side vectors is introduced.

The figure below shows the GPU acceleration performance for a 56M DOF vehicle trim body model. The same machine is used as for the stator model. The AMS eigensolution achieves a 2.1x speedup using 32 CPUs and 4 GPUs compared to a 32 CPU-only run.



For more information, see *Parallel Execution in Abaqus/Standard*.

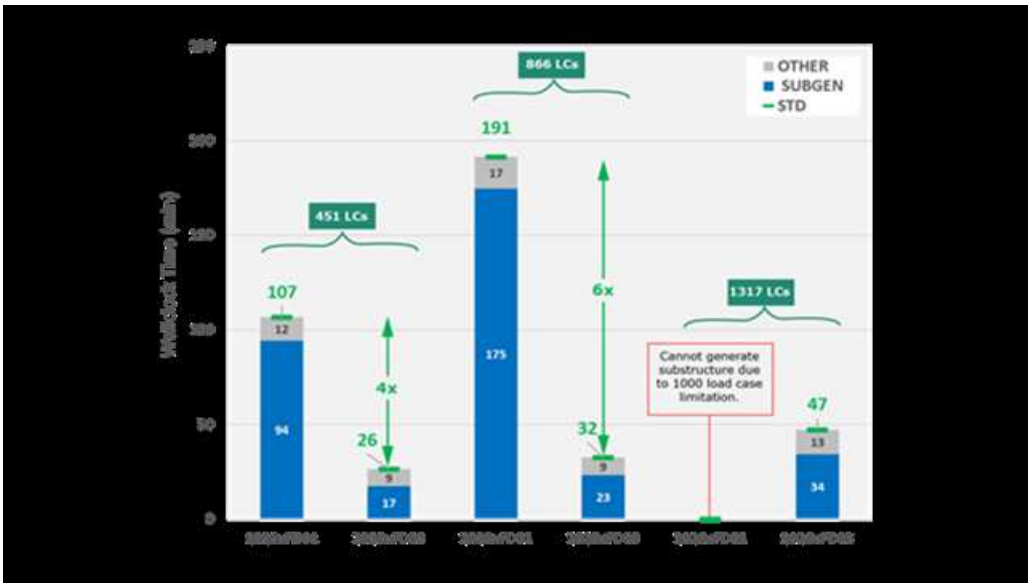
Substructure Load Case Generation

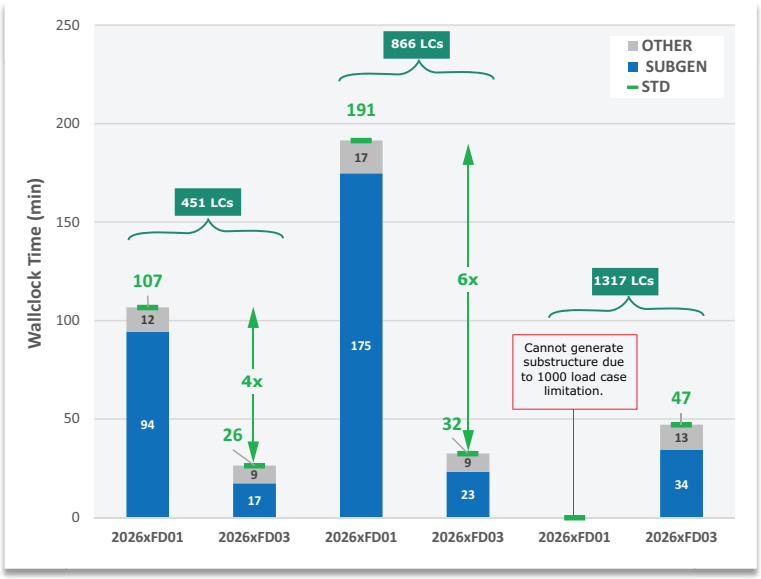
You can now generate substructures with more than 1000 load cases.

Benefits: Substructure generation analyses run faster, and you can generate all substructure load cases required in workflows using substructures.

In earlier releases, you could generate no more than 1000 load cases in a single substructure generation step.

The benchmark engine model demonstrates the enhancements. The finite element mesh has 11 M degrees of freedom. Generation of a substructure with 78 retained nodal degrees of freedom and 227 retained eigenmodes is performed for three cases with increasing numbers of substructure load cases. The analysis is run on a machine with a 64-core Emerald Rapids processor and 512 GB memory. In the figure below, STD is the total analysis time, SUBGEN (blue portions of the bars) is the time substructure generation time, and OTHER time (gray portions of the bars) includes eigenvalue extraction using the AMS eigensolver and other analysis phases.





For more information, see *Generating Substructures*.

Specifying Time-Dependent Tool Orientations and Bead Dimensions in Trajectory-Based Eigenstrain Additive Manufacturing Simulations

In a trajectory-based eigenstrain additive manufacturing simulation, you can now specify a time-dependent tool orientation and a time-dependent height and width of the deposition bead.

Benefits: You can more accurately model additive manufacturing simulations.

You specify the tool orientation and the bead dimensions as part of the event series definition used to describe the tool trajectory during the additive manufacturing simulation.

For more information, see *Eigenstrain-Based Analysis of Additive Manufacturing Processes*.

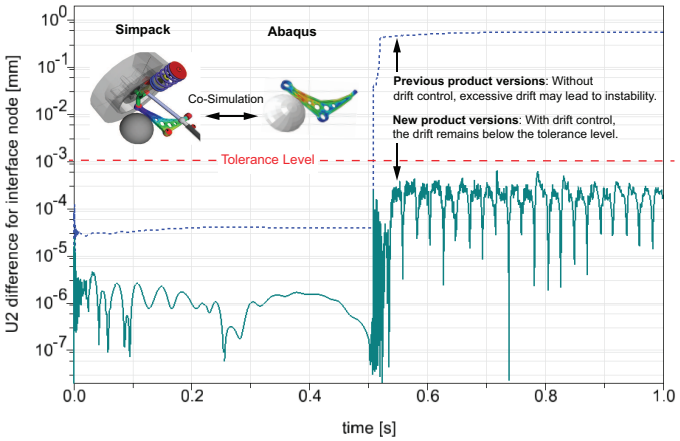
Abaqus-to-Simpack Co-Simulation

An Abaqus-to-Simpack co-simulation now computes initial interface forces to establish acceleration compliance of the transient FE and MBD subsystems. Algorithmic enhancements also minimize displacement drift.

Benefits: Setting up an Abaqus-to-Simpack co-simulation is easier, particularly when it starts from a nonresting state. You can also obtain more accurate and robust solutions.

Abaqus-to-Simpack co-simulation uses the enhanced subcycling method that ensures velocity compatibility at the interface. However, this method can lead to a displacement drift when the co-simulation uses a large coupling step size. Now, the co-simulation monitors the displacement drift during the solution phase and employs a Baumgarten type of approach to compensate for the displacement drift. In some cases, the coupling step size is also reduced automatically to keep the drift small. This leads to a more accurate solution and more robust coupling of both transient subsystems.

You must use Simpack 2027 to benefit from these enhancements.



Coupled Simulation Guide

A new Coupled Simulation Guide describes the SIMULIA Co-Simulation Services. It provides details about the SIMULIA Co-Simulation Services API and Co-Simulation Configuration file and examples of how to embed the services into your application.

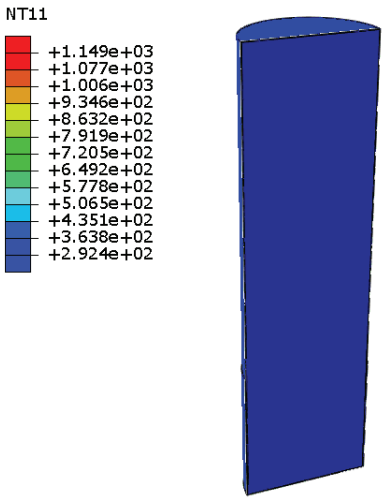
Benefits: You can more easily embed the SIMULIA Co-Simulation Services into your application.
The SIMULIA Co-Simulation Services expands the simulation capabilities of Abaqus by allowing you to perform coupled multiphysics and multiscale simulations between your application and Dassault Systèmes physics solvers and partner applications for importing fields in sequential coupled simulations, solving complex multiphysics co-simulations (for example, fluid-structure interaction), and multidomain coupling.

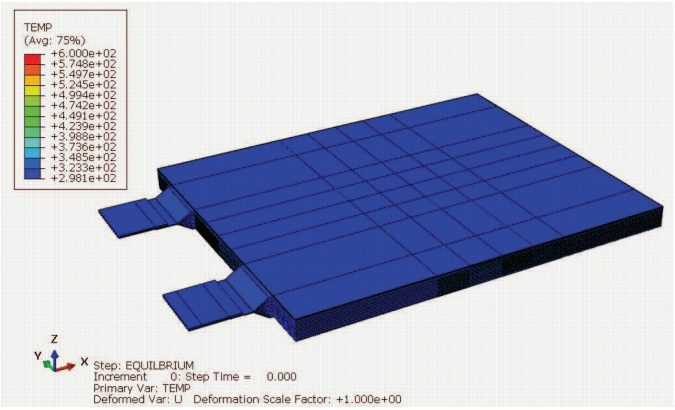
This is an advanced capability. You must have sufficient programming skills, in-depth knowledge of the Dassault Systèmes application being coupled, and familiarity with the limitations of the segregated solution methods for co-simulation. We encourage contacting customer support to discuss your project before embarking on any programming efforts.

Modeling Thermal Runaway in Lithium Ion Batteries

You can use coupled thermal-electrochemical analysis and fully coupled thermal-electrochemical-structural analysis to model thermal runaway in lithium ion batteries.

Benefits: You can now use Abaqus to more effectively assess battery safety by considering thermal runaway. Thermal runaway is a very rapid, self-feeding, uncontrolled event that releases large amounts of heat due to the exothermic reactions of the various battery components, such as the electrolyte, SEI layer, and decomposition of the lithium at the electrodes. The animations below show the rapid rise in temperature in cylindrical and pouch cells during a thermal runaway event.





For more information, see *Modeling Thermal Runaway in Batteries*.

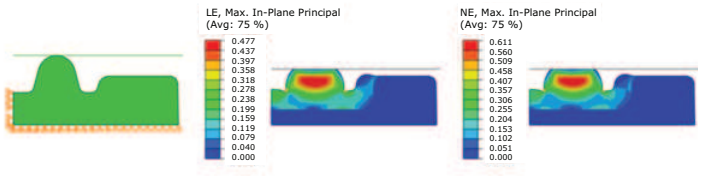
2026 FD02

Adjoint Sensitivity Analyses

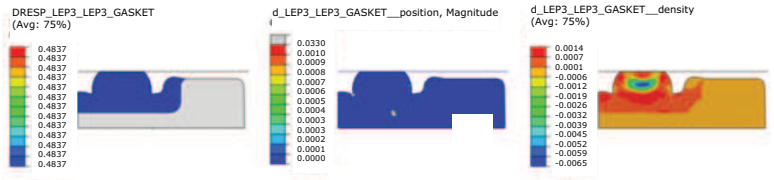
You can now specify strain, logarithmic strain, and nominal strain as design responses in adjoint design sensitivity analyses, and you can use hybrid elements in transient dynamic adjoint shape sensitivity analyses.

Benefits: You have more flexibility in design optimization workflows.
You can now design structures based on optimizing different strain measures. You can also include incompressible material response (for example, rubberlike materials) in the design domain for transient dynamic shape optimization.

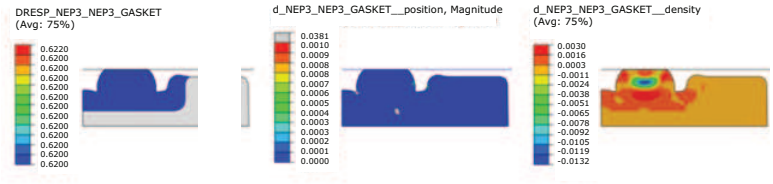
The figure below shows the loading, boundary conditions, and logarithmic and nominal principal strains for a rubber gasket sealing simulation using the plane strain assumption and hybrid elements due to nearly incompressible material response.



The figure below shows the maximum LEP3 as a design response, along with the corresponding sensitivities with respect to the nodal shape design variables and elemental topology design variables, respectively.



The figure below shows the corresponding results with the maximum NEP3 as the design response.



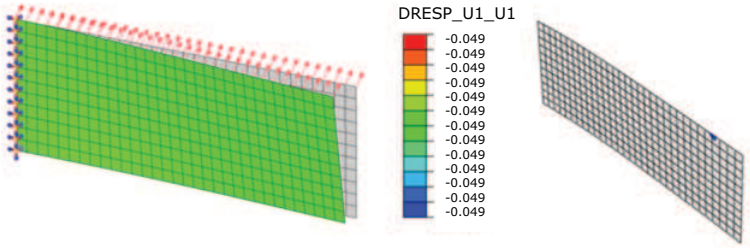
For more information, see *Specifying Design Responses in a Static or an Eigenfrequency Analysis*, *Specifying Design Responses in a Transient Dynamic Analysis*, and *Adjoint Design Sensitivity Analysis Overview*.

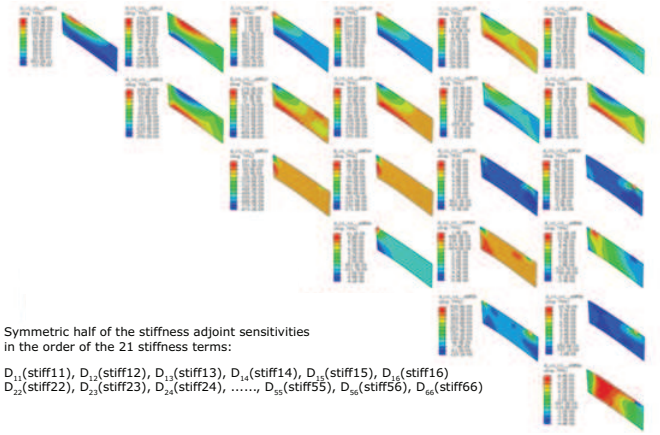
ABD Stiffness Sensitivity for Shell Elements Defined Using a General Shell Section

In Abaqus/Standard you can now specify the components of the section stiffness (ABD) matrix for shell general sections as design variables in an adjoint design sensitivity analysis.

Benefits: You can use the ABD stiffness sensitivities to optimize the design of composite shell structures. Adjoint design sensitivities with respect to the components of the section stiffness (ABD) matrix allow you to carry out generative design for the best possible overall stiffness properties for a composite shell structure.

The figure below shows the setup for an adjoint ABD design sensitivity analysis for a shell structure subjected to a single static load case and with nodal displacement, U1, of one of the nodes as the design response. Each of the 21 stiffness components of the ADB matrix for each element is considered to be a design variable. The second figure shows the adjoint sensitivities of the design response with respect to each of the 21 design variables.



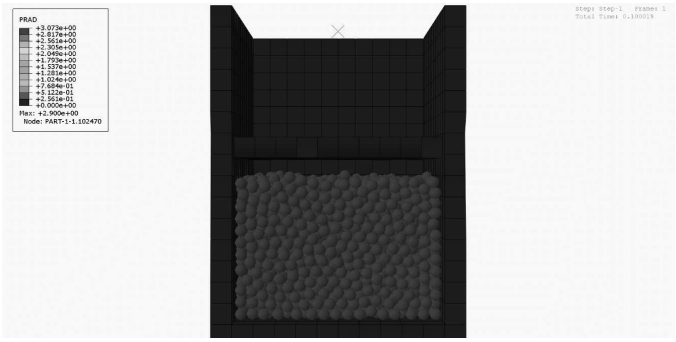


Particle Swelling for Discrete Element Analysis

Abaqus/Explicit now supports swelling of discrete particles.

Benefits: You can better understand the changes in media properties in granular media undergoing swelling. Granular media properties such as porosity depend on particle size. When the particle size changes due to swelling, media properties such as porosity also change.

The animation below shows the granular media behavior with reversible swelling that is modeled using discrete element method particles with user-defined swelling behavior.



For more information, see *Discrete Element Method*.

2026 FD01

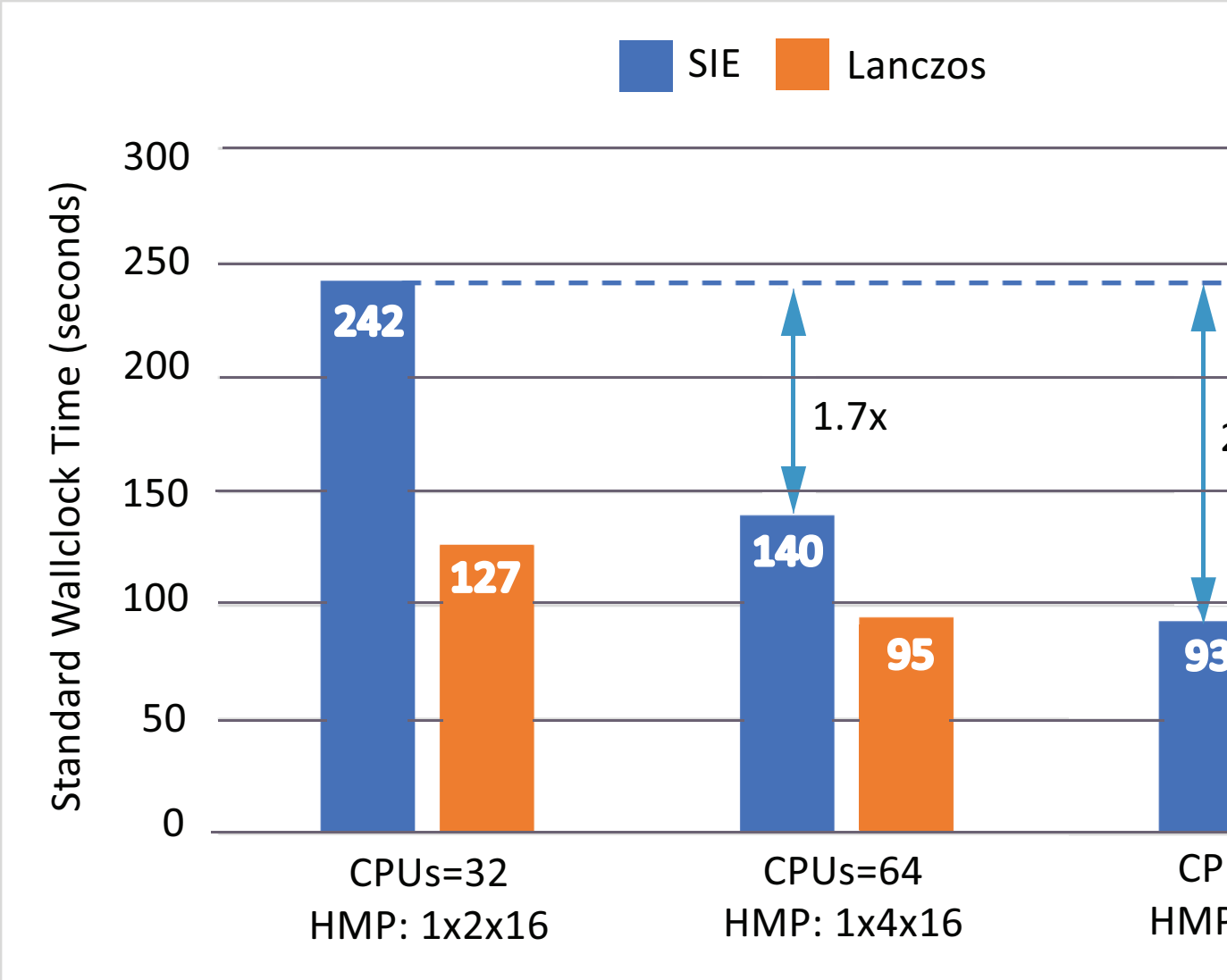
Subspace Iteration Eigensolver Functionality Enhancements

You can now perform frequency extraction analysis using MPI-based parallel execution of the subspace iteration eigensolver.

Benefits: You can see significant performance improvements for the frequency extraction of coupled structural-acoustic problems.

Frequency extraction analysis using the subspace iteration eigensolver can run in hybrid memory parallel mode for coupled structural-acoustic problems and support residual model computation and damping projection for subsequent modal analysis.

The figure below illustrates the enhanced performance of the subspace iteration eigensolver using a coupled structural-acoustic analysis with ACOUSTIC COUPLING=PROJECTION for a Neon model with 2.5 million degrees of freedom.



The analysis is run in hybrid memory parallel (HMP) mode using up to two compute nodes, each of which has an Intel Xeon 3.5 GHz Emerald Rapids processor with 2 x 32 cores and 512 GB memory.

In each of these runs, we request 50 modes up to a maximum frequency of 300 Hz, specify an acoustic range factor of 2.0, request residual modes, and turn on damping projection.

We compute a total of 112 modes (50 in the acoustic domain + 50 in the structural domain + 12 residual modes) with both the Lanczos and subspace iteration eigensolver.

The subspace iteration eigensolver exhibits a 1.7x and 2.6x speedup as we go from 32 cores (1 x 2 x 16 host configuration) to 64 cores (1 x 4 x 16 host configuration) and 128 cores (2 x 4 x 16 host configuration), respectively. In addition, the subspace iteration eigensolver is comparable to the Lanczos eigensolver on two compute nodes.

Subspace iteration also delivers good accuracy in subsequent modal analyses.

For more information, see *Natural Frequency Extraction* and **FREQUENCY*.

Performance Improvement from Alternate Contact Convergence Criteria

Abaqus/Standard now modifies contact-related convergence checks automatically when you specify an alternate convergence check.

Benefits: The solution time might be reduced.

The alternate convergence checks were introduced in an earlier release; however, in earlier releases they did not affect contact-specific convergence thresholds. Now, contact convergence is influenced when you set the CONVERGENCE CHECK parameter equal to MODERATE, RELAXED, or CONCEPT DESIGN also influence on the **CONTROLS* option.

The table below shows the effect on the total number of iterations on three contact models due to modified contact convergence checks. Each row in the table corresponds to a different setting of the convergence checks.

Model	Convergence Check Setting	Total # of iterations in earlier releases	Total # of iterations in 2026x FD01	Reduction in total # of iterations
1	STRICT (default)	857	857	No change
	MODERATE	760	732	3.7%
	RELAXED	801	766	4.5%
	CONCEPT DESIGN	784	749	4.5%
2	STRICT (default)	25,361	25,361	No change
	MODERATE	24,991	12,760	49.0%
	RELAXED	24,852	14,953	39.9%
	CONCEPT DESIGN	24,943	14,393	42.3%
3	STRICT (default)	556	556	No change
	MODERATE	494	494	0%
	RELAXED	380	380	0%
	CONCEPT DESIGN	380	380	0%

No changes to the convergence tolerance occur across versions with the default STRICT setting in effect; therefore, the total number of iterations remains the same for all models that use this setting. The impact for models with the MODERATE, RELAXED, and CONCEPT DESIGN settings is as follows:

- Small percentage reductions in the number of iterations for Model 1, with the DESIGN setting requiring the fewest iterations.
- Larger percentage reductions in the number of iterations for Model 2, with the MODERATE setting requiring the fewest iterations. This shows that further relaxation of convergence thresholds does not always lead to convergence improvement, which is somewhat counterintuitive. You might occasionally observe the STRICT setting leading to better convergence than the other settings.
- No effect on the number of iterations for Model 3, indicating that contact-specific convergence checks do not control convergence for this model.

In terms of accuracy, Model 2, which exhibits the largest impact compared to earlier releases, was examined. There were no differences in peak Mises stresses or peak contact stresses due to the changes for each of the alternate convergence check settings. For this model, both in earlier releases and in this release, the peak contact stresses were within 3% and the peak Mises stresses were within 0.1% of each other for the different settings. In general, you must ensure that solution accuracy from these alternate checks is acceptable on its own or in comparison with the default STRICT setting.

For more information, see *Modifying the Default Convergence Checks* and **CONTROLS*.

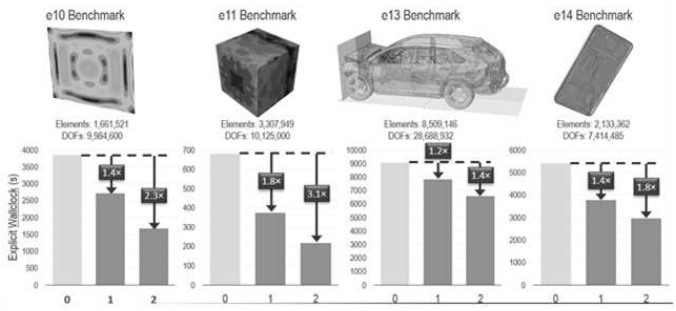
GPU Acceleration in Abaqus/Explicit

For NVIDIA GPUs on Linux systems, Abaqus/Explicit now supports GPU acceleration for a wide variety of dynamic and quasi-static structural simulations.

Benefits: Performance is significantly improved, particularly when running large models.
Abaqus/Explicit takes advantage of the compute acceleration on NVIDIA GPUs on Linux systems. The solver can be executed either on a single node with one or more GPU devices or on a multi-node cluster with multiple GPUs. The implementation emphasizes results accuracy compared to CPU-only execution.

GPU execution supports most features, with exceptions for user subroutines, CEL, ALE, and particle methods. The acceleration from GPU usage depends on the computational intensity of the model and data flow between the CPU cores and the GPU devices. Therefore, the performance benefit is workflow dependent. Typically, large models with hundreds of thousands of elements to multiple millions of elements can run significantly faster by using GPU acceleration.

The figure below shows several examples of Abaqus/Explicit GPU acceleration over a range of workflows with the addition of 1 and 2 GPUs.



The benchmarks e10 and e11 are dominated by element computation. The production level models e13 (a crash simulation) and e14_DropTest_v0 (a phone drop test) demonstrate a range of performance gains with the addition of 1 or 2 A100 GPUs to a 48-core Icelake node. On the challenging side, the crash model (whose compute cost is dominated by contact) shows a moderate performance gain, whereas the drop test (which still has a decent number of contacting parts) shows a significant gain. The other two examples show an excellent speed up that can be achieved for element computation-dominated workflows.

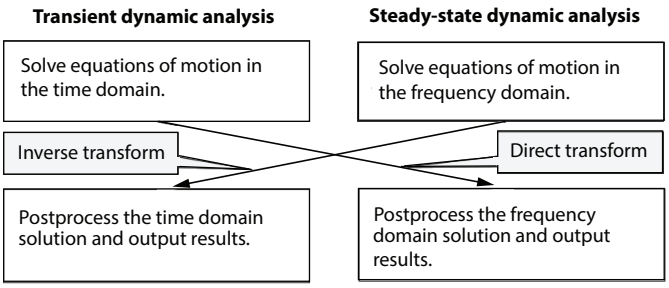
For more information, see *Parallel Execution in Abaqus/Explicit*.

Fourier Transform in Linear Dynamics Procedures

You can now use a Fourier transform in linear dynamics procedures to convert the mechanical response from the time domain into the frequency domain and vice versa.

Benefits: The new functionality improves the durability and NVH workflows.
You can now obtain a:

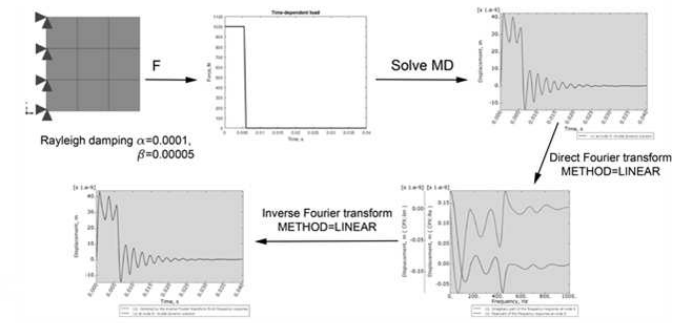
- Frequency response from a time-domain response using a direct Fourier transform.
- Time-domain response from a frequency response using an inverse Fourier transform.



Abaqus applies the Fourier transform on the modal level (to the generalized displacements). Therefore, the computational complexity of the Fourier transform is independent of the number of degrees of freedom in the model. Instead, the computational complexity of the implemented Fourier transform algorithms depends on the number of eigenmodes, time, and frequency points.

The example below illustrates validation of the direct and inverse Fourier transforms. First, a modal dynamics problem was solved in the time domain. Then, a direct Fourier transform was applied that results in the frequency

response (in the frequency domain). Finally, an inverse Fourier transform was applied that restores the original motion in the time domain. The plots shown demonstrate horizontal displacement of the point at which the force is applied.



For more information, see *About Dynamic Analysis Procedures*.

SPH Performance Improvements and Memory Usage Reduction When Modeling Liquids

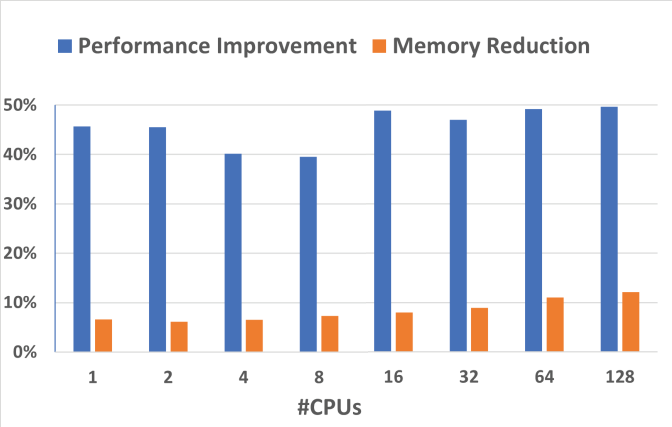
You can now conduct SPH simulations in Abaqus/Explicit with a new particle-to-particle pairwise interaction-based algorithm.

Benefits: Performance is improved and memory usage is reduced for SPH simulation in liquid modeling.

In earlier releases of Abaqus/Explicit, the SPH method used an algorithm based on a pseudo element type architecture: a SPH particle and its surrounding cloud particles within the smoothing length range formed a pseudo element. All element-level calculations were based on the particle interactions within this pseudo element. This code architecture, while inline with other Lagrangian element types in Abaqus/Explicit, does not take advantage of the particle-to-particle pairwise-interaction nature of the SPH method and results in suboptimal performance and excessive memory usage, particularly when modeling liquids.

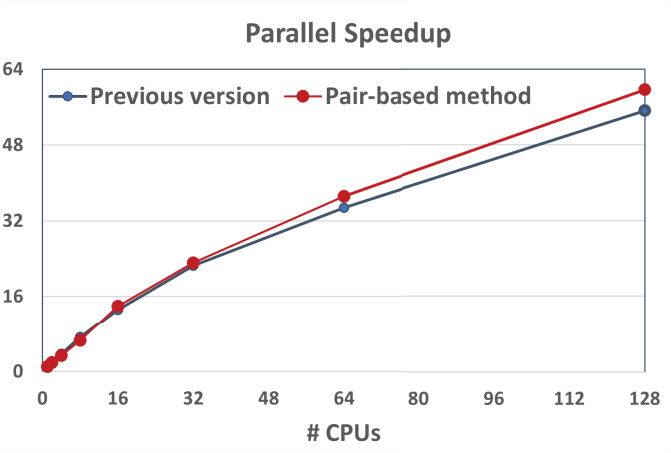
Now, you can use a new SPH element solving technology using a particle-to-particle pairwise-interaction based solution. The neighboring particle search and domain repartition algorithms are rearchitected to further improve performance. This new algorithm is adopted by default for liquid materials modeled with the Mie-Grüneisen equation of state (*EOS, TYPE=USUP) when the pressure is dependent of the internal energy (only the reference speed of sound and the slope of the UsUp curve are defined for the material).

The example below illustrates the performance improvements and memory reduction achieved using the particle pair-based solution technology for a tank sloshing analysis with different number of CPUs. The first figure shows the performance improvement and memory usage reduction compared to the previous pseudo-element-based approach across different numbers of CPUs.



The performance improvement ranges from ~39% to ~49%, while the memory usage reduction ranges from ~6% to ~12%. The performance improvement and memory usage reduction for any given SPH model might vary depending on the presence of other features such as contact, deformable elements, etc.

The second figure shows the parallel speedup improvement across different numbers of CPUs. The parallel speedups are noticeably better for the new particle pair–based SPH method when the number of CPUs is larger.



You cannot use this new feature in combination with the improved SPH boundary behavior (*SPH SURFACE BEHAVIOR) or when the *EOS, TYPE=USUP material definition includes tensile failure (*TENSILE FAILURE).

For more information, see *Smoothed Particle Hydrodynamics*.

Enhanced GPU Support for Modal Dynamic Solvers

Abaqus now delivers faster, more efficient GPU acceleration for mode-based steady-state and transient modal dynamic analyses.

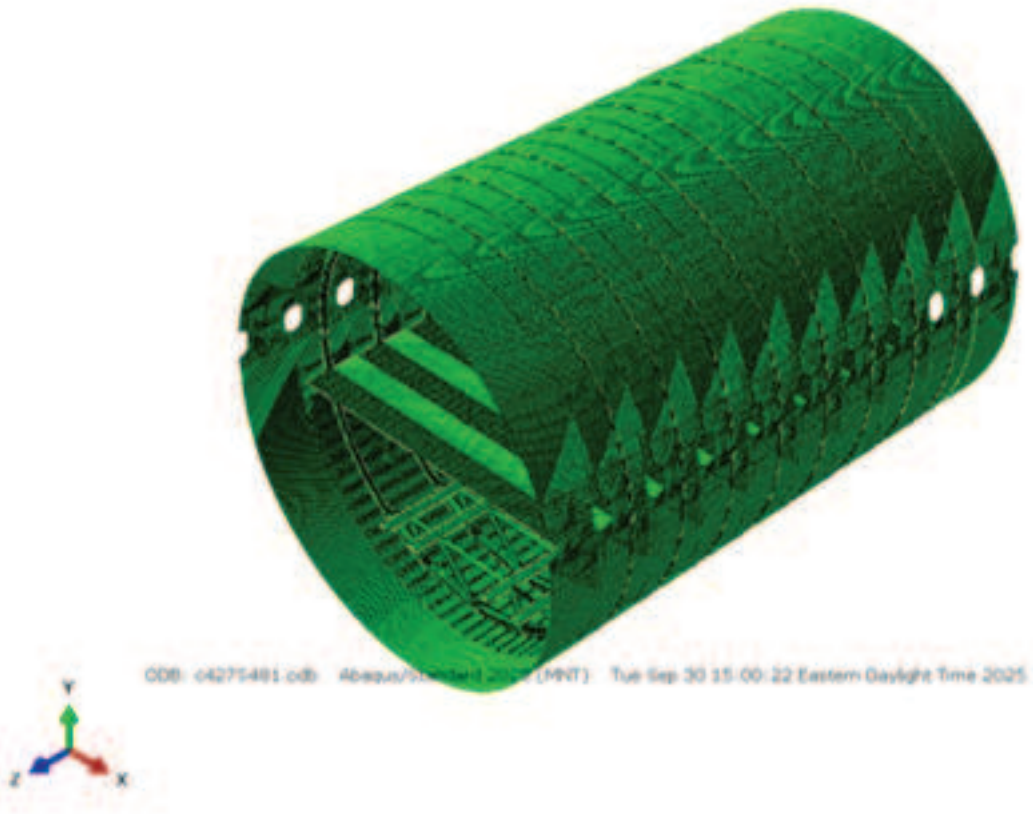
Benefits: These solver enhancements dramatically improve performance and scalability for linear dynamic simulations on GPU-enabled systems, providing faster, more efficient analysis of large-scale models and complex dynamic scenarios across steady-state and transient workflows

GPU support for modal analysis solvers in Abaqus has been significantly enhanced to deliver faster and more scalable solutions. Performance of the multi-GPU Conventional mode-based steady-state dynamics solver has been improved substantially. A newly implemented GPU-based Spectral solver extends GPU support for the mode-based steady-state dynamic analysis. In addition, a new GPU-based State Space solver enables efficient

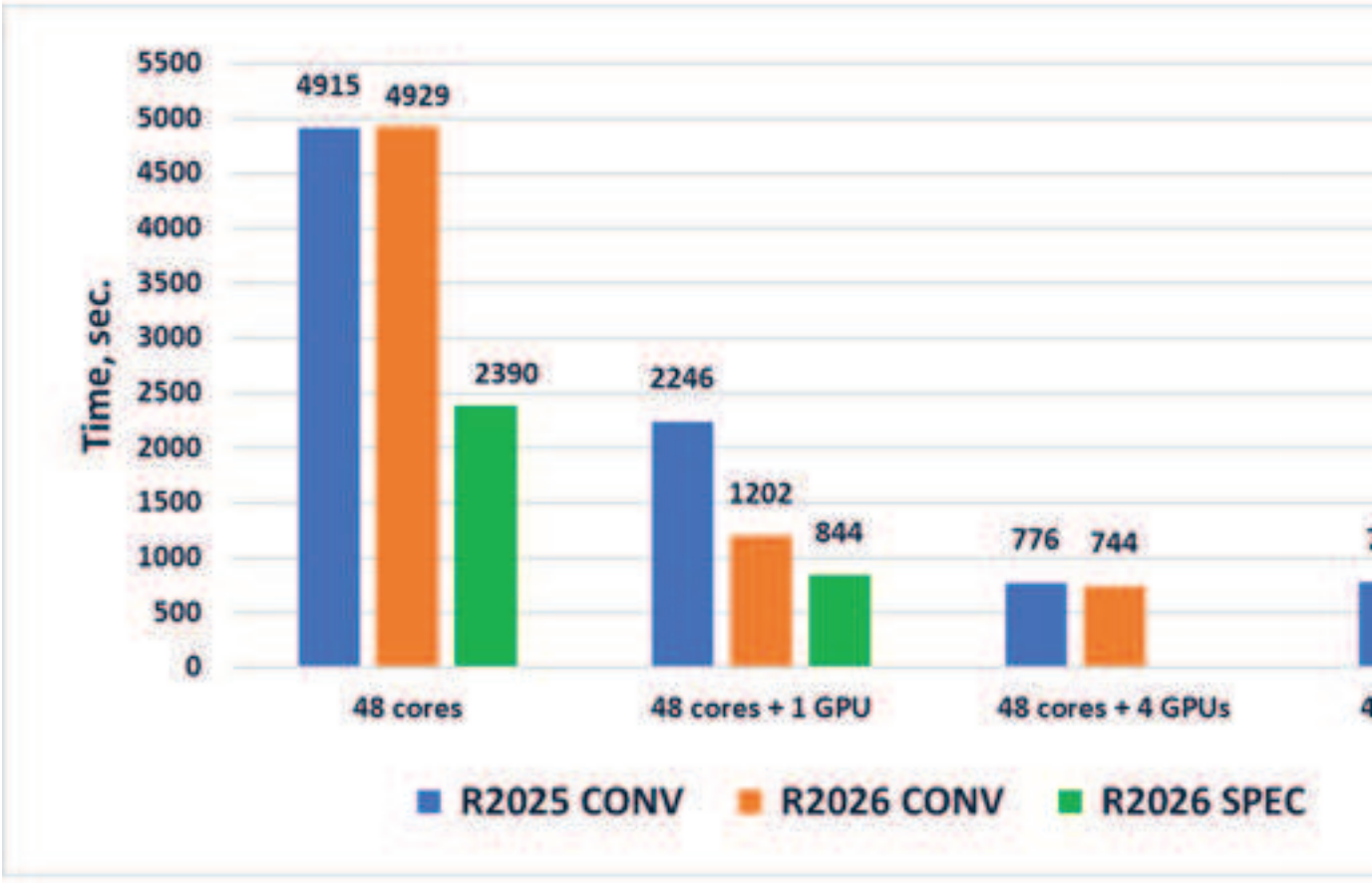
transient modal dynamic simulations. These improvements unlock substantial performance gains, especially for large-scale models and complex dynamic scenarios.

Example 1. Frequency Response Analysis – Fuselage Model

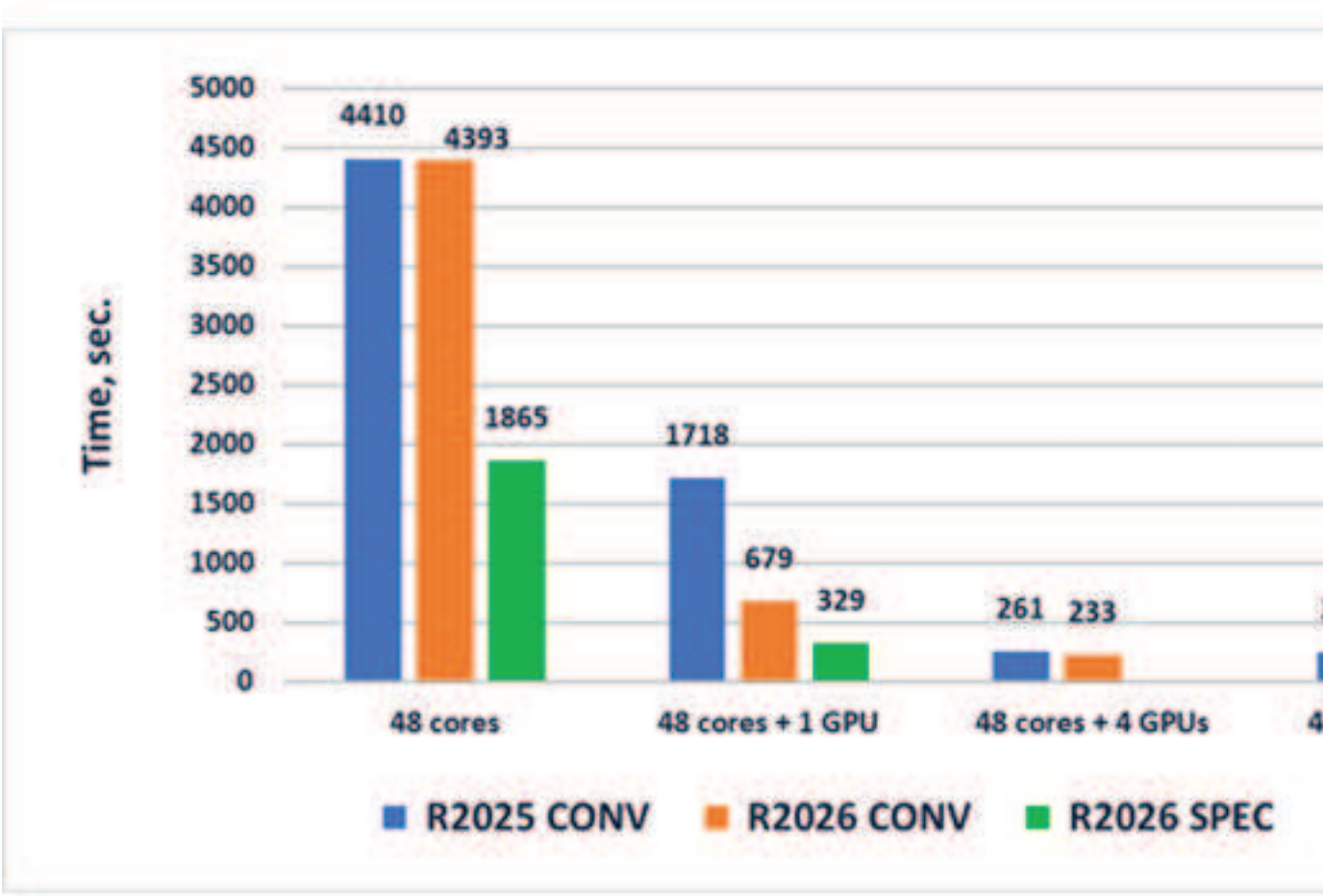
- 48.3 million degrees of freedom
- Structural material damping
- 1000 frequency points
- 120 load cases
- 21,327 eigenmodes



This example highlights the solver's ability to efficiently handle high-resolution modal steady-state dynamic (SSD) simulations. The figure below shows the total analysis time for the fuselage model.



The figure below shows the modal SSD solver time for the fuselage model.



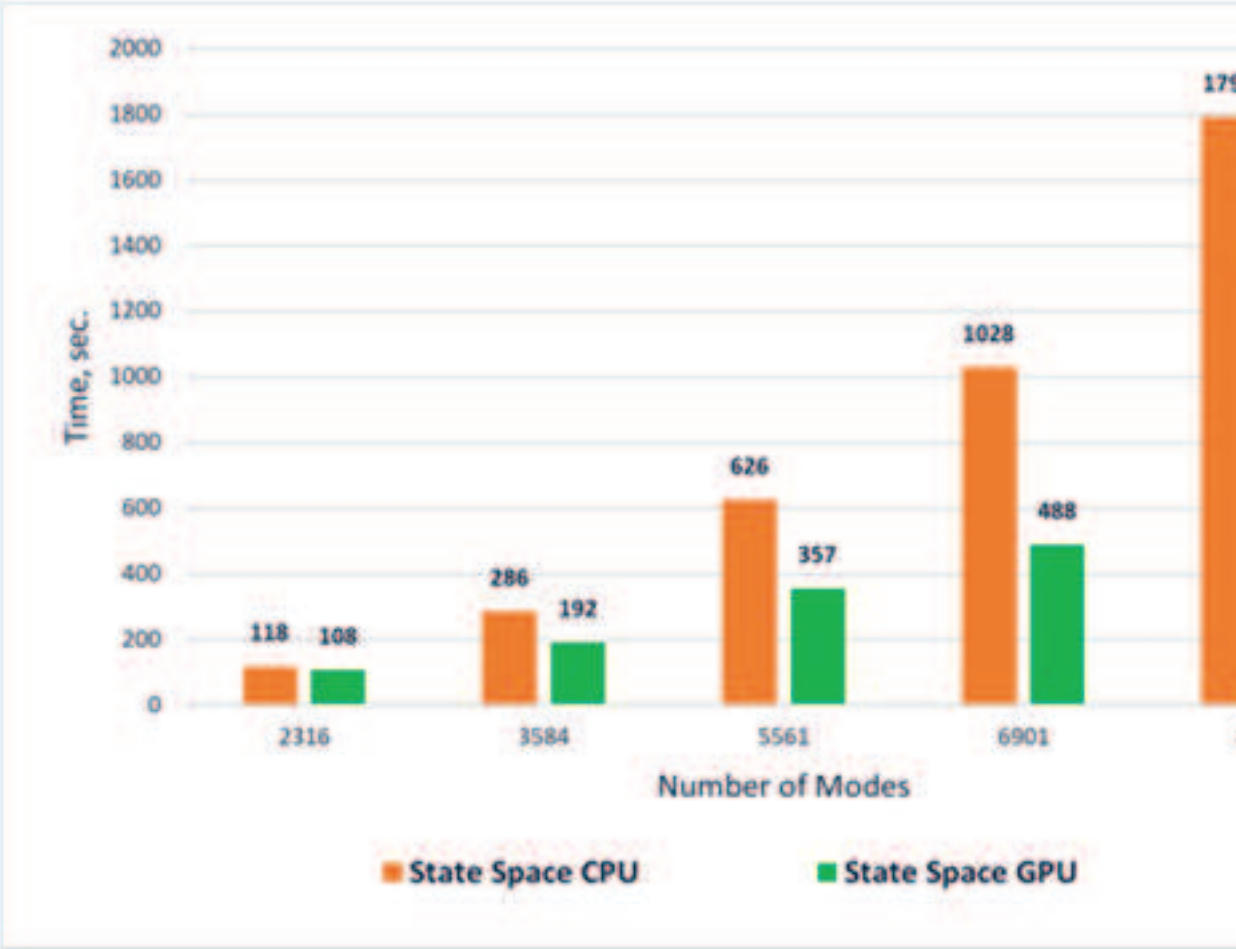
Example 2. Transient Modal Dynamic Analysis – Body-in-White Model

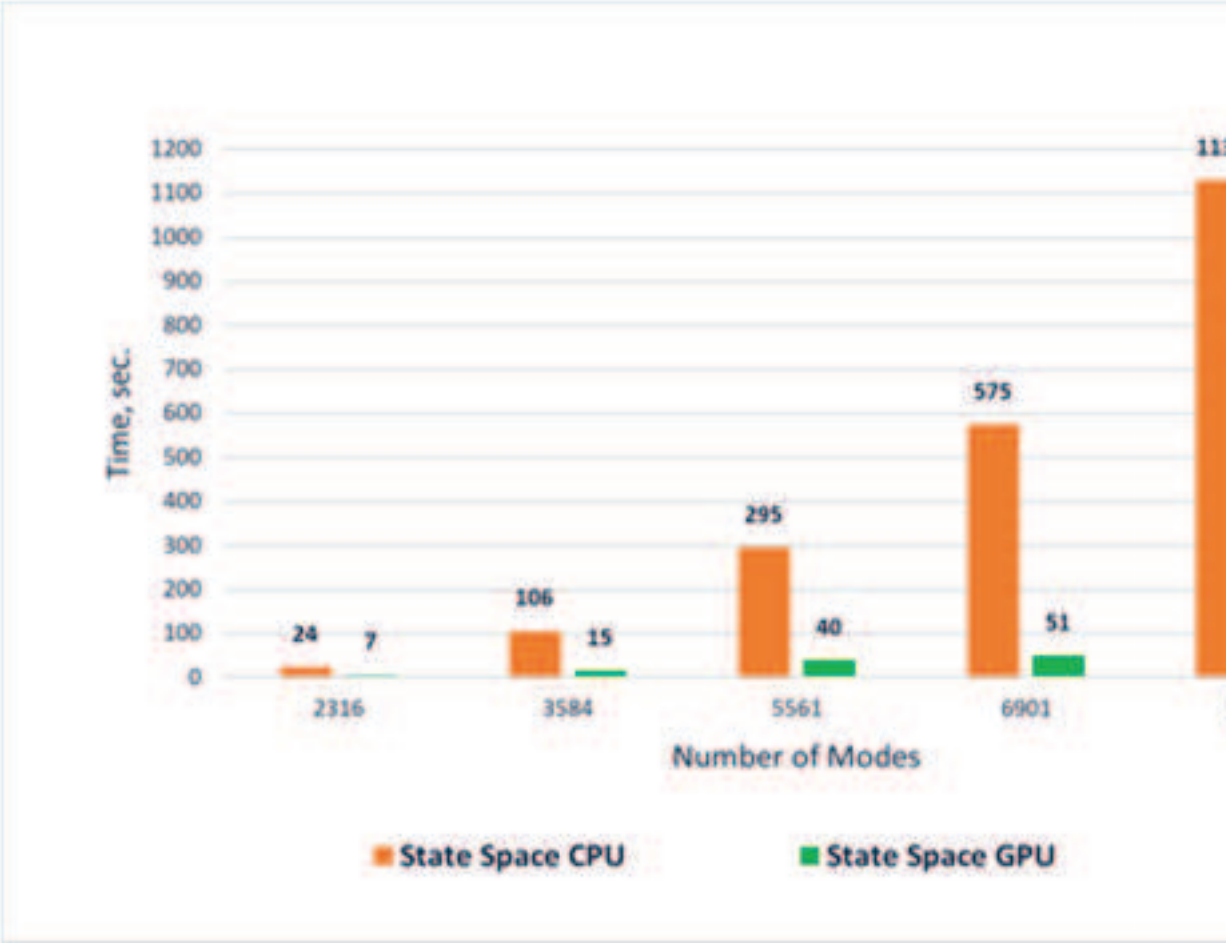
- 780,000 degrees of freedom
- Material viscous damping
- 10,000 time points



This example demonstrates the performance improvements achieved using the GPU-based State Space solver for transient modal dynamic analysis for several simulations with different number of eigenmodes used for the modal approximation of the solution.

The figure below shows the total analysis time for the body-in-white model.


2000
1800
1600
1400
1200
1000
800
600
400
200
0
Time, sec.
118
108
286
192
626
357
1028
488
2316
3584
5561
6901
Number of Modes
State Space CPU
State Space GPU



For more information, see *Parallel Execution in Abaqus/Standard*.

New Parallel Architecture for the SIMULIA Co-Simulation Services

The SIMULIA Co-Simulation Services (CSS) supports a new distributed memory parallel architecture to enhance performance, capacity, and scalability.

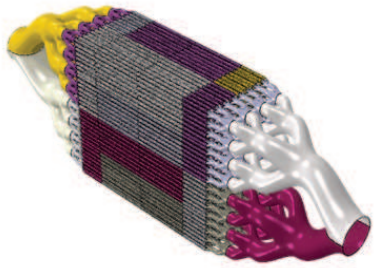
Benefits: The new CSS architecture supports multiple data communication channels and mapping units to maximize the computing resource utilization. You will see greatest performance benefits for large volumetric coupled simulation.

The parallel CSS architecture is a scalable architecture for high-performance computing. The parallel architecture utilizes multiple compute ranks for field transfer and mapping. The new architecture:

- supports large volumetric coupled simulations which the serial CSS architecture was not able to handle before; and
- provides parallel field transfer and mapping.

The figure below shows the structural model of the heat exchanger and the parallel mapping domains. The temperature field was computed by a conjugate heat transfer analysis using the SIMULIA Navier-Stokes solver

(available in the Fluid Scenario Creation, and subsequently imported into a Abaqus/Standard thermal-stress analysis.



The CFD model has close to 40 million cells and the structural model has more than 13 million elements. The table below provides the speedup of the mapper's neighborhood search and mapping during the field transfer for different number of RGrids. An RGrid is a geometric partition of the interface region on which a mapper unit operates on. Significant speedup is observed by engaging most of the client's processors to map. This speedup further reduces the partner client's CPU idle time during a co-simulation while waiting for updated results.

The CSS-API can be embedded in either serial or parallel; both serial and parallel implementations can cooperate. Furthermore, a serial client might delegate mapping to the parallel client, thus maximizing the use of compute resources.

This new capability requires that the clients use the latest CSS libraries. Please refer to *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* to find third-party compatibility with Abaqus 2026 FD01 or later.

Table 1: Mapper Speedup for Heat Exchanger Model

CPUs	RGrids	Search Time (s)	Search Speedup	Map Time (s)	Map Speedup
64	1	162.1	1	30.0	1
64	8	73.8	2	3.9	7.7
64	16	23.5	6.9	1.9	15.8
64	32	9.8	16.5	0.94	31.9
64	64	4.3	37.7	0.47	63.8

For more information, see *About Co-Simulation* and *General Capability for Importing External Fields*.

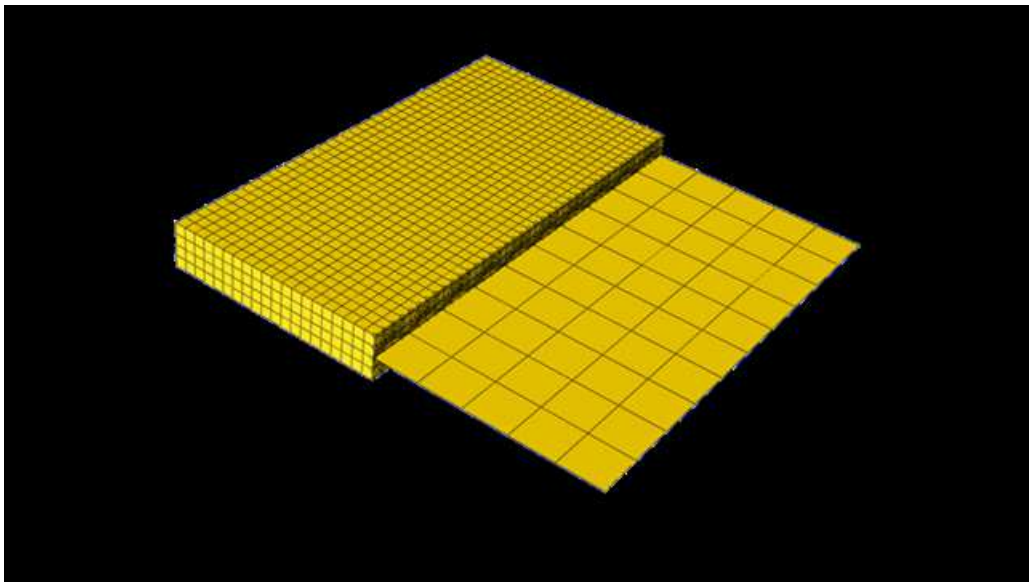
Beam-to-Beam and Solid-to-Solid Submodeling Using the General Submodeling Interface

General submodel now supports beam-to-beam and solid-to-solid submodeling in addition to beam-to-solid, beam-to-shell, shell-to-shell, and shell-to solid submodeling including temperature, pore pressure and acoustic pressure degrees of freedom.

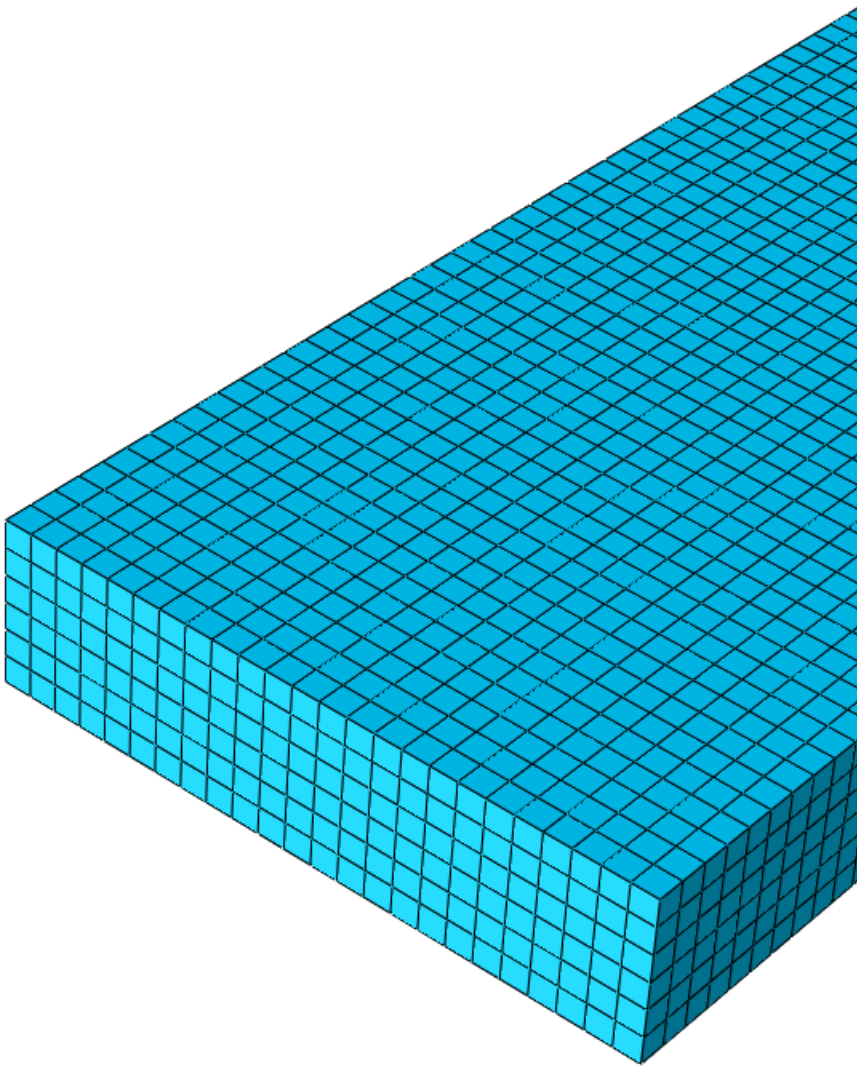
Benefits: You can now perform submodeling with a broader mix of element and analysis types between global and submodel analyses.

The example shown below demonstrates use of solid-to-solid and shell-to-solid submodeling together.

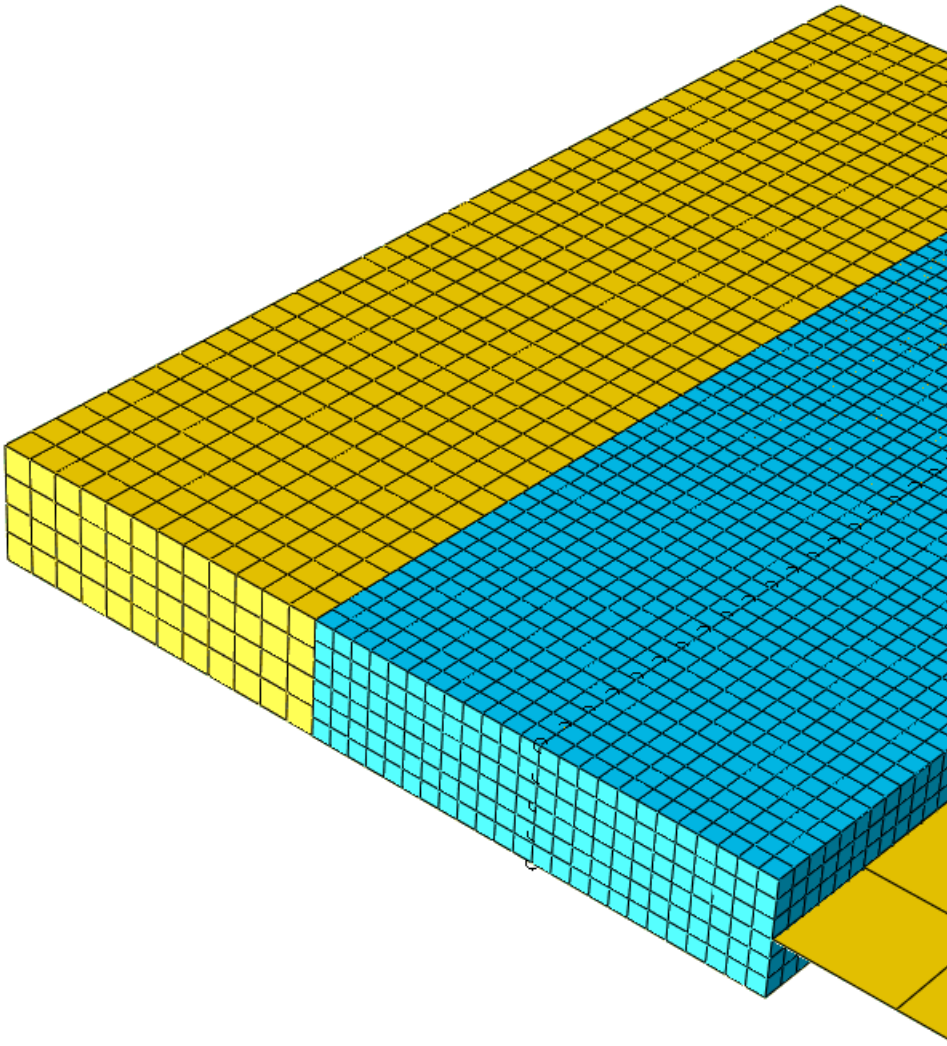
The figure below shows the global model of a plate modeled with solid elements in one region and shell elements in another region with shell-to-solid coupling connecting the two regions.



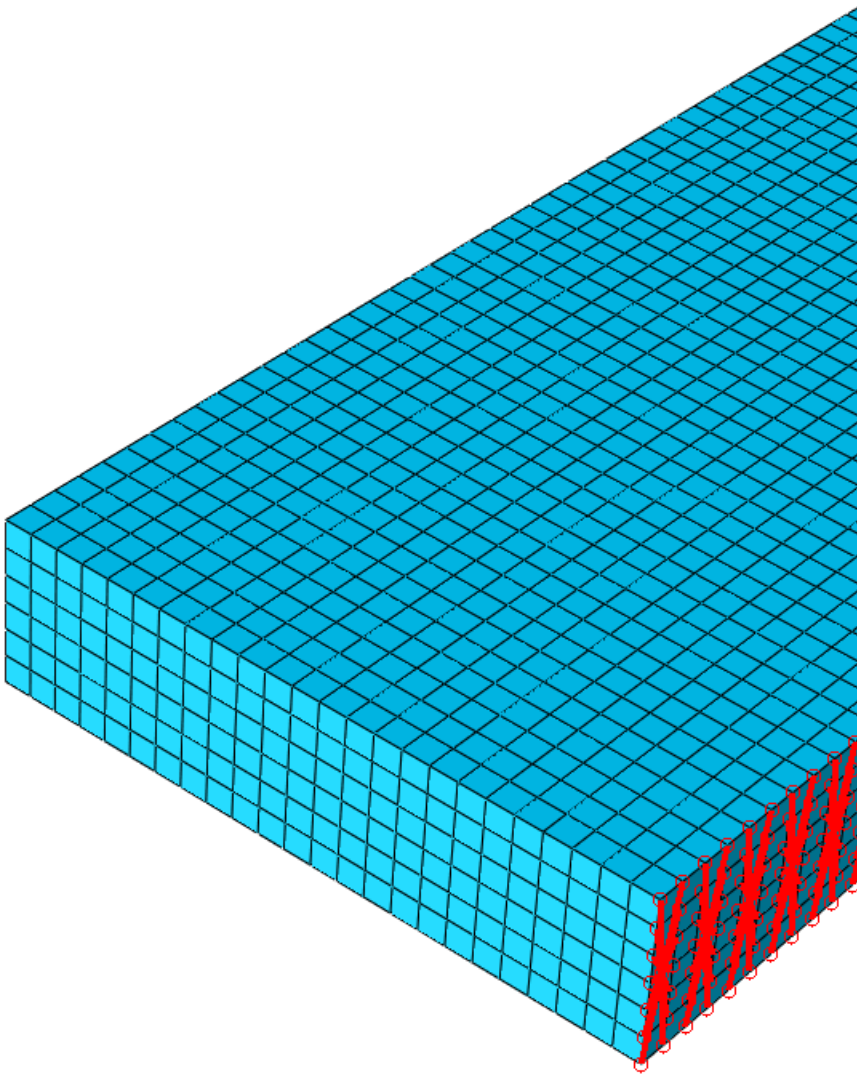
The figure below shows the submodel meshed with solid elements that occupies the middle portion of the global model as shown in the subsequent figure. The interface on one side is shell-to-solid, and the interface on the other side is solid-to-solid.



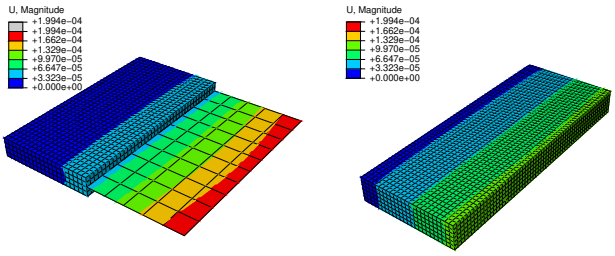
The figure below shows the global and submodel superimposed.



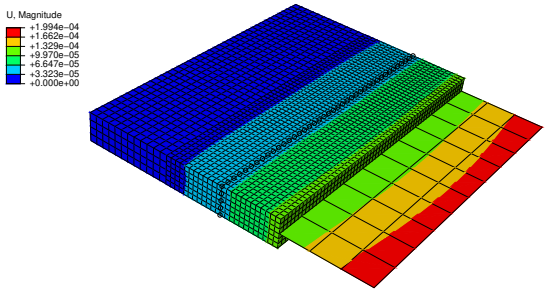
The figure below shows the 30 internally generated distributing couplings used to impose submodel conditions at the submodel cut surface that corresponds to the shell element region of the global model. Displacements and rotations at these internally generated reference nodes are imposed from the interpolated results within the shell elements of the global model. The other submodel cut surface corresponds to solid-to-solid submodeling and displacements at the nodes of that submodel cut surface are imposed based on interpolation of results within solid elements of the global model.



The figure below compares the magnitude of displacement solution for the global model (left side) and the submodel (right side).

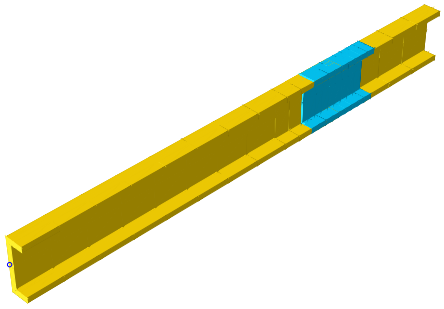


The figure below shows overlay displacement magnitude contour plots for the global and submodel of the example demonstrating shell-to-solid and solid-to-solid submodeling.

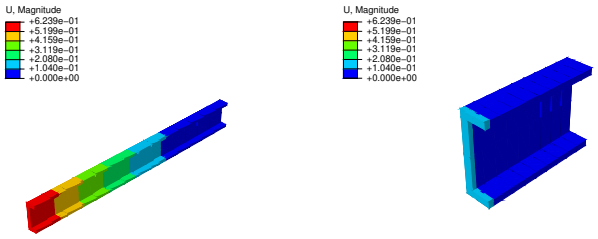


The next example demonstrates beam-to-beam submodeling for a small section near the root of a cantilever beam.

The figure below shows the global and submodel superimposed, with the global model colored gold and the submodel colored blue. Beam elements are used for both the models.



The figure below shows the plot of the displacement solution for the global model and the submodel. Beam cross section rendering is used for both these figures.



The keyword to define the submodel cut surface has been moved from history data to model data. You can no longer select the degrees of freedoms on which to apply the submodel conditions, you must apply submodel conditions to all the primary degrees of freedoms of the nodes on the submodel cut surface.

For more information, see *General Submodeling*.

2026 GA

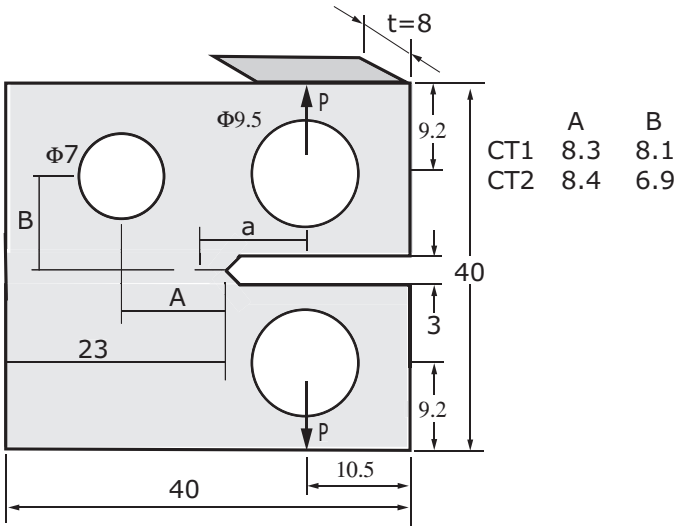
Fatigue Crack Growth with Adaptive Remeshing

You can now use fatigue crack growth with adaptive remeshing in Abaqus/Standard.

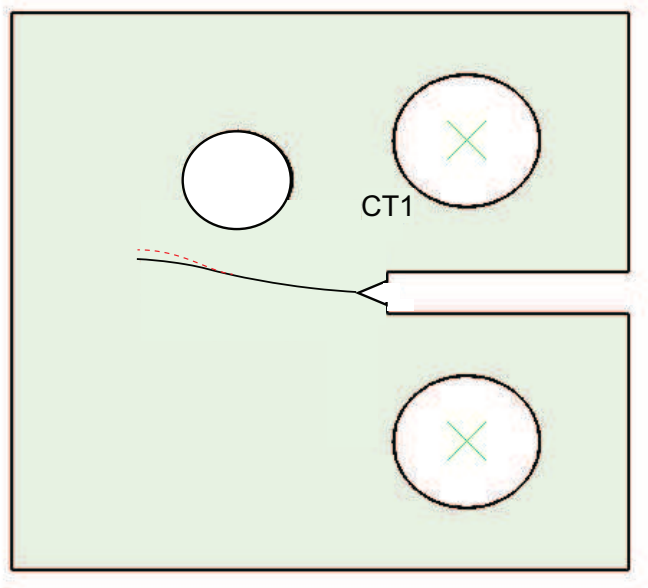
Benefits: This functionality combines the fracture/fatigue technology in Abaqus/Standard with a tetrahedral element-based adaptive remeshing technology to predict crack growth under cyclic loading conditions. The crack growth rate is based on the Paris Law, which is driven by the classical contour integral method for computing energy release rates or stress intensity factors at the crack front.

The overall workflow is semi-automated and involves automatic execution of a series of successive Abaqus analyses with progressively longer crack lengths, without any direct user intervention. The tetrahedral element–based adaptive remeshing technology is used to automatically create a new mesh after each increment of crack growth. The new mesh conforms with the updated crack front. The capability is applicable only for linear elastic fatigue crack growth analysis; therefore, state mapping is not necessary between any two successive increments of crack growth.

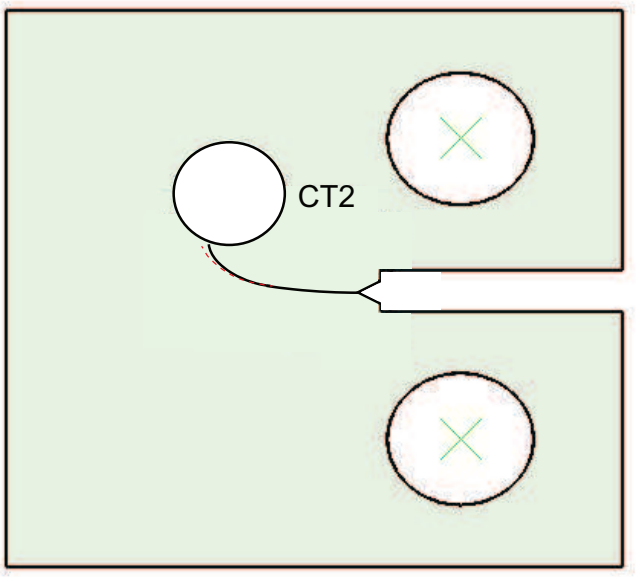
The figure below shows a modified compact tension specimen, following Miranda et al. (2003), with an additional hole that is machined to curve the crack propagation path. The dimensions are in mm. The crack either propagates into the hole or deflects away from it, depending on the hole's location.



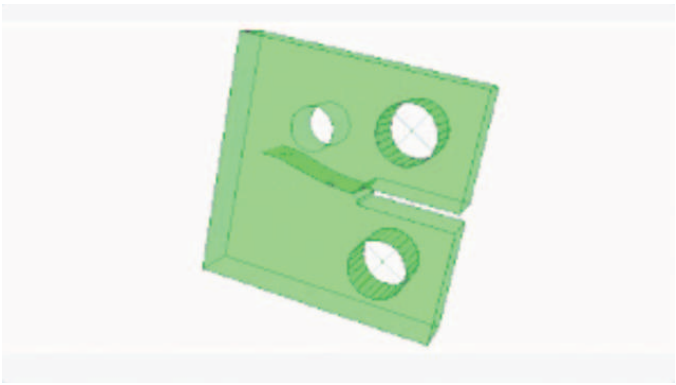
The figure below shows the crack paths predicted by Abaqus overlaid on the experimental measurements for the modified compact tension CT1 specimen.



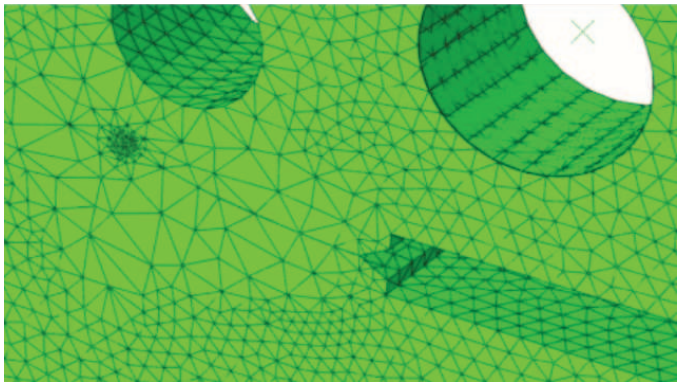
The figure below shows the crack paths predicted by Abaqus overlaid on the experimental measurements for the modified compact tension CT2 specimen.



For the CT1 specimen, the animation below shows how the crack initially propagates toward the hole and then deflects away from it as it gets near to the hole.



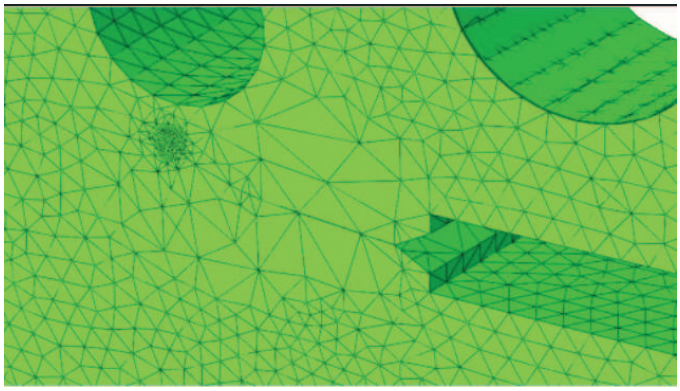
The following animation shows a local view:



For the CT2 specimen, the animation below shows how the crack propagates toward the hole and finally sinks into the hole.



The following animation shows a local view:



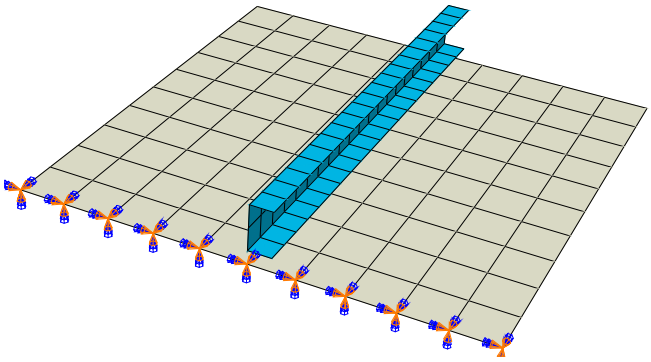
For more information, see *Fatigue Crack Propagation with Adaptive Remeshing*, **ADAPTIVE REMESH*, **ADAPTIVE REMESH RETAIN*, **CONTOUR INTEGRAL*, **FATIGUE*, and **FRACTURE CRITERION*.

Shell-to-Solid and Shell-to-Shell Submodeling Using the General Submodeling Interface

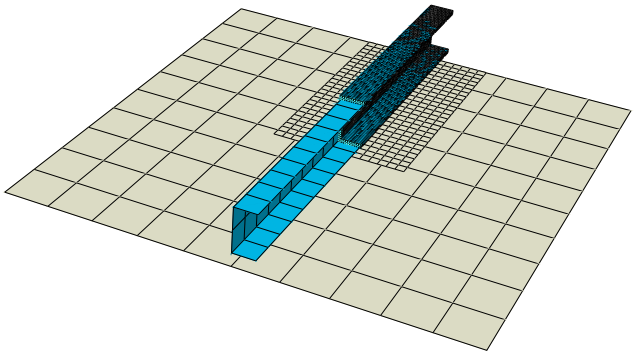
The general submodel interface now supports shell-to-shell and shell-to-solid submodeling in addition to the beam-to-solid and beam-to-shell submodeling cases supported in earlier releases.

Benefits: You can now perform submodeling analyses with a broader mix of element types between the global and submodel analyses.
The following example demonstrates shell-to-shell and shell-to-solid submodeling in a single submodel analysis.

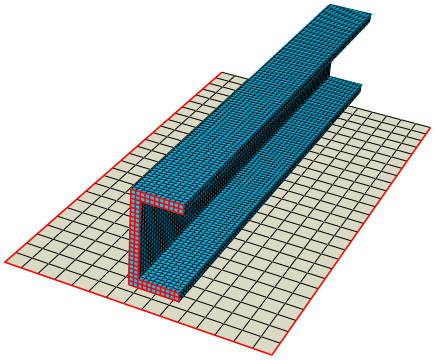
The figure below shows the global model of a stiffener tied to a plate, with the stiffener and plate meshed with shell elements.



The figure below shows a submodel superimposed with the global model.

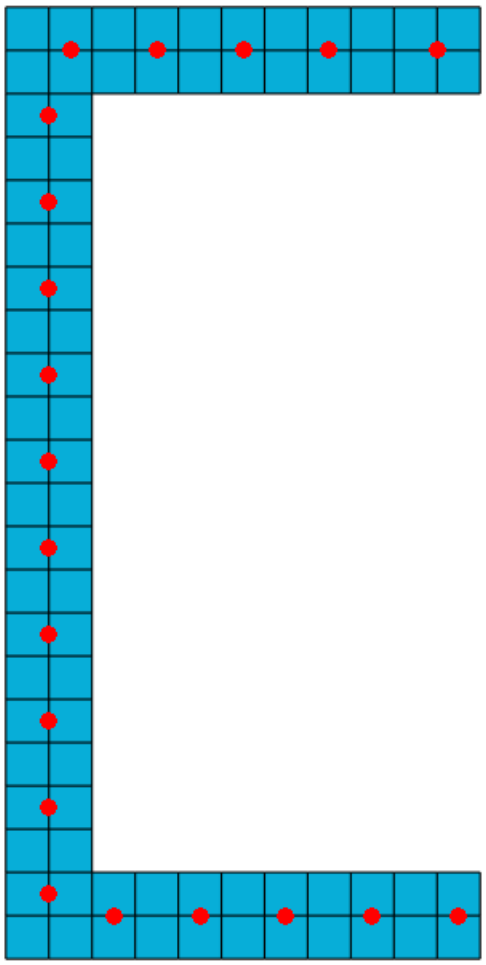


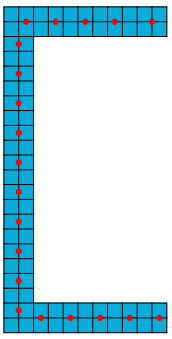
The figure below shows the submodel alone and highlights the submodel cut where results from the global model drive the submodel. The plate has a finer shell element mesh for the submodel than in the global model, and the stiffener is meshed with solid elements in the submodel and shell elements in the global model.



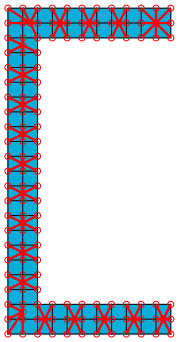
The method used to impose conditions from the global model on a local model at a shell-to-shell submodel cut involves interpolation from the global model results. The method used to enforce conditions from the global model on the local model at a shell-to-solid submodel cut is more complex but easy to use. In this case, Abaqus/Standard internally generates distributing couplings with cloud nodes of solid elements on the cut surface of the submodel. The number of reference nodes and their positions on the submodel interface are determined by an automated algorithm. Displacements and rotations at reference nodes of the distributing couplings are interpolated from results of shell nodes of the global model. Each of the driven solid nodes belongs to a single distributing coupling.

The figure below shows the positioning of the internally generated reference nodes in this example.

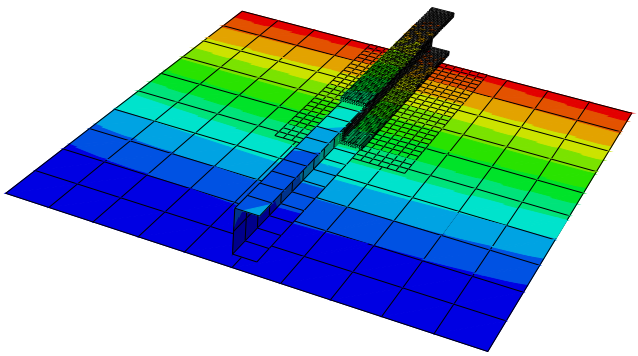




The figure below highlights the connectivity of the internally generated distributing couplings for the shell-to-solid submodel.



The figure below shows the superimposed displacement results for the global model and the submodel with good correlation of results for the global model and the submodel.



For more information, see *General Submodeling*.

2025 FD04

Internal Generation of Parameter Table Types, Property Table Types, and Event Series Types for Additive Manufacturing Simulations

For most additive manufacturing simulations, Abaqus now automatically includes the definitions for the "built-in" special-purpose parameter table types, property table types, and event series types in your model.

Benefits: It is much easier to input the data needed for additive manufacturing simulations.

The data model for the special-purpose additive manufacturing technique is based on using special table collections and event series. In earlier releases, you had to explicitly include the definitions for the "built-in" parameter table types, property table types, and event series types specific to the special-purpose additive manufacturing simulations in your input file. Now, Abaqus generates the information provided by these definitions internally.

The one exception is when you output your simulation in SIM format. In this case you must still provide the definitions explicitly in the input file.

For more information, see *Special-Purpose Techniques for Additive Manufacturing*.

2025 FD03

MPI-Based Parallel Execution of the Subspace Iteration and Lanczos Eigensolvers

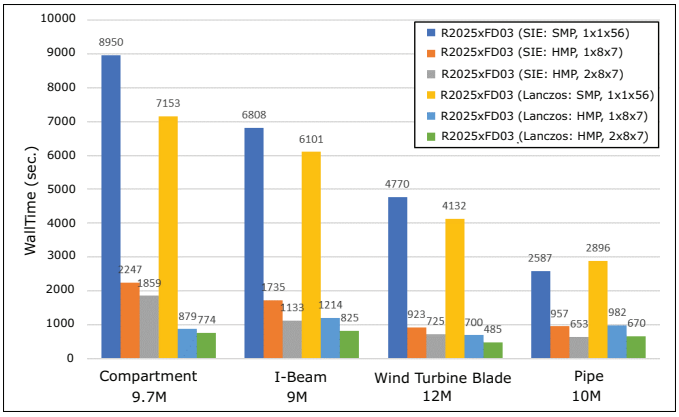
You can now use MPI-based parallel execution of the subspace iteration eigensolver for frequency extraction and buckling analyses and MPI-based parallel execution of the Lanczos eigensolver for buckling analysis.

Benefits: You can solve larger buckling and frequency extraction problems than previously possible and obtain better performance by leveraging multiple MPI ranks across multiple compute nodes.

You can now perform frequency extraction and buckling analysis using the subspace iteration eigensolver in hybrid-memory-parallel (HMP) mode. By utilizing both MPI and thread parallelism within a single compute node, as well as across multiple compute nodes on HPC hardware, the subspace iteration eigensolver significantly improves the performance of both types of analysis.

In addition, buckling analysis using the Lanczos eigensolver can be executed in HMP mode, delivering significantly enhanced performance on HPC hardware with many-core systems. Frequency extraction with the Lanczos eigensolver was introduced in earlier releases.

The figure below illustrates the performance improvement of Abaqus/Standard using the subspace iteration eigensolver and Lanczos eigensolver in 2025 FD03 for four benchmark models extracting 50 buckling eigenmodes. These four models were run in hybrid mode using up to two compute nodes, each of which has an Intel Xeon 2.6 GHz Ice Lake processor with 2 × 28 cores and 512 GB memory.



For the four benchmark models, both eigensolvers shows superior performance in HMP mode (1 × 8 × 7) compared to the shared-memory-parallel (SMP) mode configuration (1 × 1 × 56) and good parallel performance using two compute nodes (2 × 8 × 7). Abaqus/Standard automatically switches to HMP execution of supported procedures for analyses running with at least 32 CPU cores on Linux when no layout is specified.

For more information, see *Parallel Execution in Abaqus/Standard* and *Natural Frequency Extraction*.

GPU AMG Iterative Solver

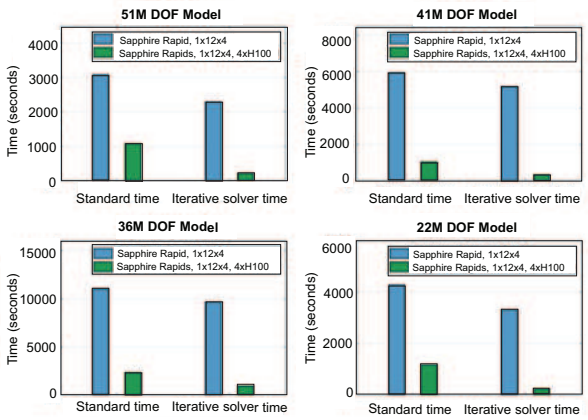
The iterative solver now supports execution across multiple NVIDIA GPUs. It can solve both symmetric and unsymmetric systems of equations without constraints involving Lagrange multipliers.

Benefits: You can now accelerate the iterative sparse solver with multiple GPUs enabling scalability that improves performance and removes model size limitations associated with the single-GPU version.

Unlike the GPU direct solver, which offloads compute-intensive operations to the GPU, the GPU iterative solver runs entirely within GPU memory.

Sufficient GPU memory is critical for successful execution. The memory requirement primarily depends on the number of nonzero entries in the matrix operator, which in turn is largely determined the element type and specific Abaqus/Standard options used (for example, **KINEMATIC COUPLING* and **EQUATION*). In general, when using linear 8-node brick elements, the peak GPU memory usage can be estimated approximately as (× 80 × 12 × *number-of-DOFs* × 1E-9 +) GB, where typically ranges from 2.5 to 4, depending on the AMG smoother used, and accounts for additional temporary workspace needed for handling multiple batch jobs during the factorization-based smoother setup.

Benchmark results comparing the CPU and GPU iterative solver performance across four models are presented below. The CPU solver was executed using a 1 × 12 × 4 configuration (Sapphire Rapids with 48 CPU cores in a single compute node running as 12 MPI ranks of 4 threads each), while the GPU solver added four H100 GPUs to these same CPU cores. The 51M DOF model is the "Medium Engine" version of the s4 engine block benchmark run with the unsymmetric iterative solver. The 41M DOF model is the s9 fuselage benchmark. The 36M DOF and 22M DOF models are static customer powertrain and vehicle frame benchmarks, respectively. Across all cases, the differences in “Iterative solver time” between the CPU and GPU solver are greater than the differences observed in overall “Standard time: because only the iterative solver computations are GPU accelerated. Several of these benchmark models cannot be run on a single GPU due to memory requirements.



For more information, see *Parallel Execution in Abaqus/Standard*.

New Parameter Table Type and Property Table Type References

You can use new parameter table type and property table type references for modeling electrochemical processes in a battery.

Benefits: You can now perform battery multiphysics simulations more easily in Abaqus/Standard.

ABQ_EChemPET_Electrode_Particle_IBVUser	Model the electrochemical process in a battery by specifying a user-defined Butler-Volmer current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDBV`.
ABQ_EChemPET_Electrode_Particle_IDissUser	Model the electrochemical process in a battery by specifying a user-defined electrode dissolution current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDDISS`.
ABQ_EChemPET_Electrode_Particle_ILPLUser	Model the electrochemical process in a battery by specifying a user-defined plated lithium current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDLPL`.
ABQ_EChemPET_Electrode_Particle_ISEIReformUser	Model the electrochemical process in a battery by specifying a user-defined Solid-Electrolyte Interface reformation current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDSEIREFORM`.
ABQ_EChemPET_Electrode_Particle_ISEIUser	Model the electrochemical process in a battery by specifying a user-defined Solid-Electrolyte Interface current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDSEIREFORM`.
ABQ_EChemPET_Electrolyte_ChargeDischarge	Model the electrochemical process in a liquid electrolyte battery by specifying a table of values that identify whether a given step in an analysis is a charging or discharging step.

For more information, see *ABQ_EChemPET_Electrode_Particle_IBVUser*, *ABQ_EChemPET_Electrode_Particle_IDissUser*, *ABQ_EChemPET_Electrode_Particle_ILPLUser*, *ABQ_EChemPET_Electrode_Particle_ISEIReformUser*, *ABQ_EChemPET_Electrode_Particle_ISEIUser*, and *ABQ_EChemPET_Electrolyte_ChargeDischarge*.

Importing all Element Sets from the Previous Analysis

You can now easily import all element sets in the previous analysis that have a section definition that is supported for import.

Benefits: You can now import all elements in the analysis model without explicitly listing the element sets to be imported.
The new, default input method imports elements once from all importable sections defined in a previous analysis. It also supports repositioning of the imported elements and importing from multiple previous analyses.

For more information, see *About Transferring Results between Abaqus Analyses* and **IMPORT*.

2025 FD02

Adjoint Fatigue Design Responses and Sensitivities

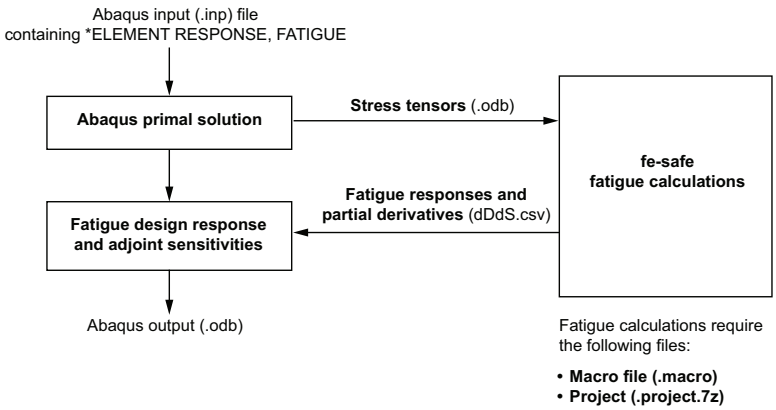
You can obtain adjoint fatigue design responses and corresponding sensitivities utilizing a combined Abaqus and fe-safe workflow.

Benefits: You can carry out Abaqus- and fe-safe-based design optimization accounting for fatigue design responses.
The results from an adjoint fatigue design sensitivity analysis for a model with shell and solid continuum elements are shown below. The analysis includes two static load cases where the shell elements are subjected to pressure loads. The stresses for these load cases are then carried over to drive the fe-safe fatigue calculations.

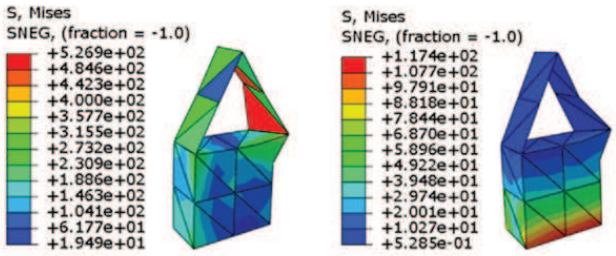
The design variables are mixed and include the following types:

- Shell element thickness (for sizing sensitivities)
- Nodal position (for shape/bead sensitivities)
- Relative material density (that scales the constitutive material properties and the stress relaxation for topology sensitivities)

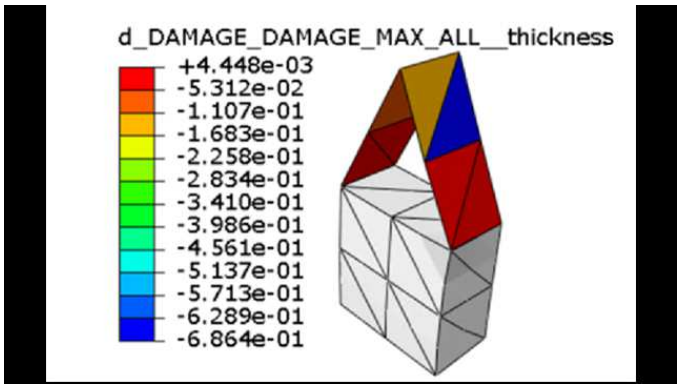
The figure below illustrates the overall combined Abaqus fe-safe workflow for fatigue design responses and corresponding adjoint sensitivities.



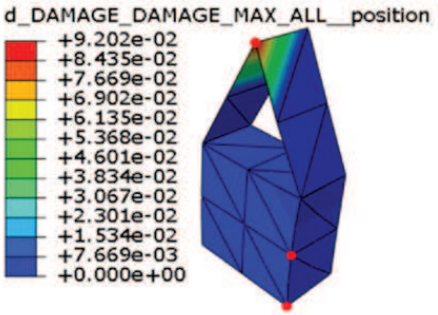
The figure below shows contour plots of the Mises equivalent stress corresponding to the two load cases in the Abaqus static analysis.



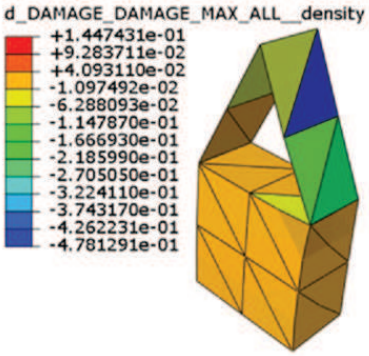
The figure below shows a contour plot of the adjoint design sensitivity of the DAMAGE design response with respect to the shell thickness.



The figure below shows a contour plot of the adjoint design sensitivity of the DAMAGE design response with respect to the nodal positions.



The figure below shows a contour plot of the adjoint design sensitivity of the DAMAGE design response with respect to relative material density.



For more information, see *Adjoint Design Sensitivity Analysis Overview*.

ABQ_EChemPET_Electrode_FluidVF_BrugExp Parameter Table Type Reference

You can use the new parameter table type reference **ABQ_EChemPET_Electrode_FluidVF_BrugExp** to specify the Bruggeman exponents to be used for computing the effective ionic diffusivity and ionic conductivity of the fluid phase in a battery.

Benefits: This feature expands the capabilities of Abaqus/Standard.

For more information, see *ABQ_EChemPET_Electrode_FluidVF_BrugExp*.

ABQ_EChemPET_Electrode_SolidVF_BrugExp Parameter Table Type Reference

You can use the new parameter table type reference **ABQ_EChemPET_Electrode_SolidVF_BrugExp** to specify the Bruggeman exponents to be used for computing the effective electrical conductivity of the solid phase in a battery.

Benefits: This feature expands the capabilities of Abaqus/Standard.

For more information, see *ABQ_EChemPET_Electrode_SolidVF_BrugExp*.

Eigenvalue Extraction Results Postprocessing

You can use a restart analysis to extract the nodal and element variables from the natural frequency extraction data.

Benefits: You can reduce the output from large natural frequency extractions and better control how results are structured by generating tailored output using postprocessing restart analyses.
For large models that need additional output from the natural frequency extraction step, using a restart analysis can improve the computational performance of the output phase and allows you to tailor output results by generating multiple, smaller files. You can select certain eigenmodes for the output or generate eigenstress, eigenstrain, or similar field output for a specific portion of the model (using an element or node set), which allows you to overcome hardware constraints (such as limited disk space on the disk). Abaqus/Standard extracts the eigenmodes from the original frequency extraction analysis database and calculates the values of the output variables without recomputing the eigenmodes and eigenfrequencies. However, you cannot use this capability to output energy and energy-related variables.

For more information, see *Natural Frequency Extraction*.

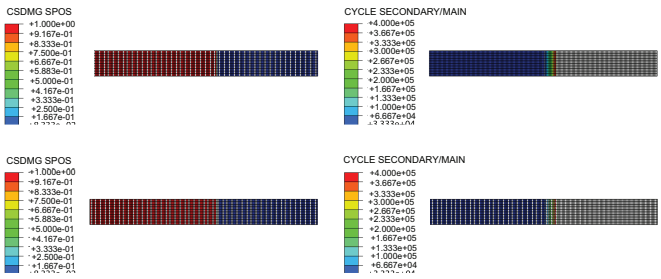
2025 FD01

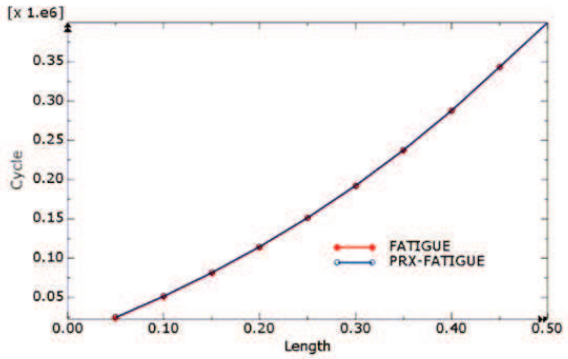
PRX-Fatigue Criterion

You can use the PRX-Fatigue criterion to model delamination propagation under cyclic loading in Abaqus/Standard.

Benefits: The PRX-Fatigue criterion offers smoother delamination fronts and faster solution times due to its better convergence characteristics.

The following figures show results from a linear elastic fatigue crack growth simulation in a single leg bend specimen, where the fracture energy release rates are computed using either the traditional-VCCT technique (top) or the PRX-VCCT technique (bottom). The results obtained by using the two different approaches agree very well with one another.





For more information, see *Crack Propagation Analysis*, *Fatigue Crack Growth Laws for Linear Elastic Materials*, **FRACTURE CRITERION*, and *Debonding behavior of a single leg bending specimen*.

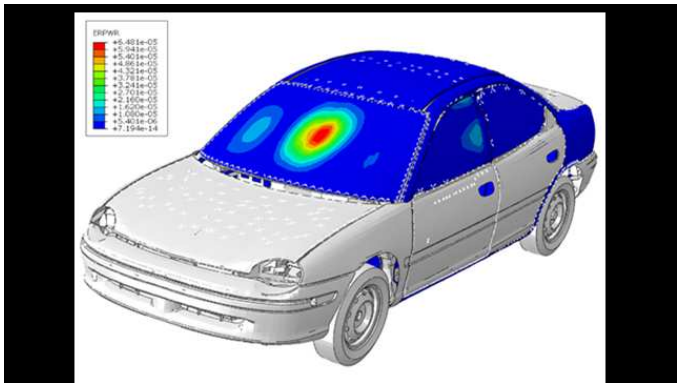
Equivalent Radiated Power Output in Steady-State Dynamic Procedures

You can request output for the equivalent radiated power, equivalent radiated power density, total equivalent radiated power, and the equivalent radiated acoustic pressure in all steady-state dynamic analysis procedures.

Benefits: The new output variables enable you to calculate these quantities when the model contains no acoustic elements or when you need to compute an equivalent radiated power on a structural element-based surface that has no contact with acoustic elements.

Output variable ERPWR represents the equivalent power that is radiated from the tributary area of a node on the element-based structural surface, while ERPWRDEN is an intensive representation of equivalent radiated power density (that is, it is normalized by the node's tributary area), which might yield more meaningful results for cases with nonuniform meshes. You can use output variable ALLERPWR to calculate the equivalent radiated power emitted by a panel (represented by a structural element-based surface). You can use output variable ERPAC to obtain the levels of the equivalent acoustic pressure on the radiating element-based structural surface without employing acoustic elements.

The image below shows a contour plot of the equivalent radiated power (PRPWR) for the Neon model. This model is used for illustrative purposes only and is courtesy of the Public Finite Element Model Archive of the National Crash Analysis Center at George Washington University.



For more information, see *Mode-Based Steady-State Dynamic Analysis* and **EQUIVALENT RADIATED SURFACE PROPERTIES*.

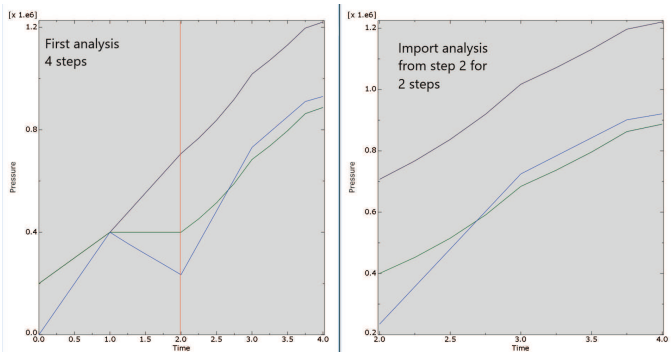
Importing the Fluid Cavity State

You can import the state of a fluid cavity bounded by a surface in any combination of Abaqus/Standard and Abaqus/Explicit import analyses.

Benefits: The transfer of the state of the reference node establishes initial equilibrium of the fluid cavity in the import analysis, and it provides consistent initial conditions for the continuation of the analysis.

The nodal state at the fluid cavity reference node (which represents the pressure inside the fluid cavity) is imported when you redefine the fluid cavity and its reference node, import or redefine the surface of the cavity, or specify import of the material state.

The example below illustrates import of pressure of several fluid cavities in an Abaqus/Explicit to Abaqus/Explicit import analysis. The left image shows the history of the fluid cavity pressures in a single analysis with four steps. The right image shows the results for the same model when you import the fluid cavity at the end of the second step.



For more information, see *About Transferring Results between Abaqus Analyses*.

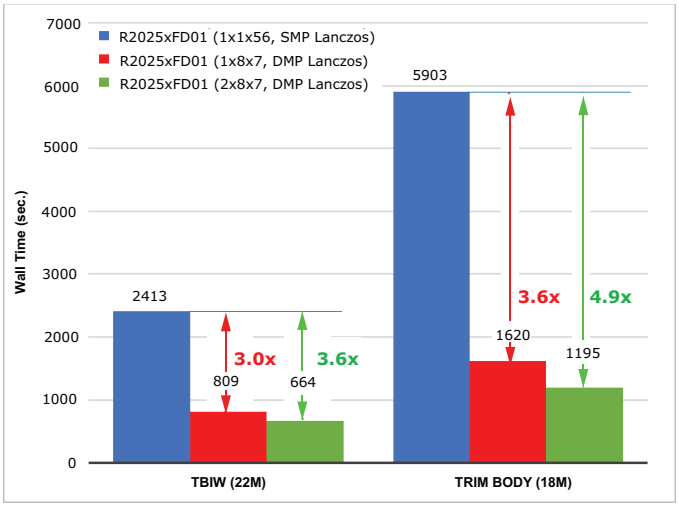
MPI-Based Parallel Execution of Lanczos Eigensolver for Coupled Structural-Acoustic Analysis

You can use MPI-based parallel execution of the Lanczos eigensolver for coupled structural-acoustic analyses.

Benefits: Frequency extraction analysis using the Lanczos eigensolver in a hybrid mode runs significantly faster and can solve large problems by utilizing multiple compute nodes for coupled structural-acoustic problems.

By leveraging both MPI and thread parallelism within a single compute node, as well as across multiple compute nodes on an HPC cluster, the improved Lanczos eigensolver significantly enhances performance and capacity for fully coupled structural-acoustic analysis workflows.

For example, the figure below illustrates the performance improvement of Abaqus/Standard using the Lanczos eigensolver in 2025 FD01 for two benchmark models with acoustic cavities.



The models are run in a hybrid mode using up to two compute nodes, each of which has an Intel Xeon 2.6 GHz Ice Lake processor with 2 × 28 cores and 512 GB memory. Running in hybrid mode yields improved performance as this is a more optimal job layout compared to SMP on these compute nodes. Additional benefits are demonstrated by the ability to leverage processor cores and memory of multiple compute nodes, which was not previously possible.

The first model is a Total Body in White (TBIW) model with 22 million degrees of freedom (DOFs) extracting 113 eigenmodes, and the second model is a trim body model with 18 million DOFs extracting 179 eigenmodes. The reference data is set to the SMP Lanczos run in 2025 FD01.

The job for the first model in a hybrid mode shows 3.0× speedup compared to the job in SMP mode on a single compute node and shows 3.6× speedup on two compute nodes. The job for the second model in a hybrid mode shows a 3.6× speedup compared to the SMP job on a single node and a 4.9× speedup on two compute nodes.

For more information, see *Parallel Execution in Abaqus/Standard* and *Natural Frequency Extraction*.

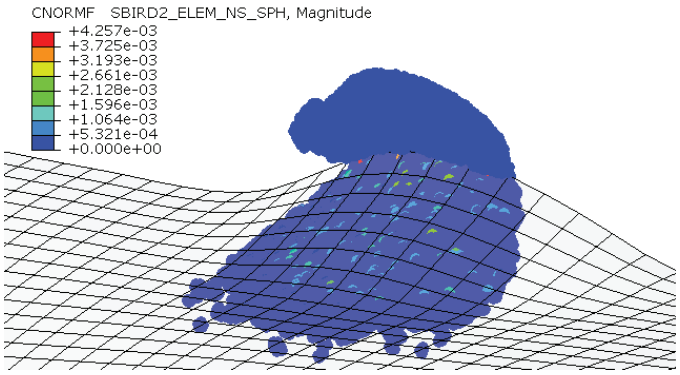
Contact Output for SPH Particles in Parallel Execution

You can request contact output for SPH particles in parallel analyses in Abaqus/Explicit.

Benefits: Supporting contact output for SPH particles in parallel analysis makes it easier to assess the simulation results.

In earlier releases, contact output for SPH particles was available only in serial execution, limiting the full assessment of SPH simulation results in parallel execution runs. You can now request contact output for SPH particles in analyses run with DMP execution.

The figure below shows the contact output CNORMF for SPH particles in a bird impact analysis for a parallel run using 8 CPUs.



For more information, see *Smoothed Particle Hydrodynamics*.

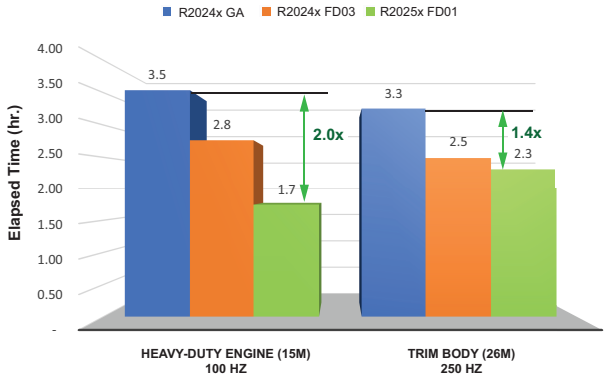
AMS Full Recovery Performance Improvement

Performance for the full recovery of eigenmodes and substructure modes for frequency extraction analysis and substructure generation analysis using the AMS eigensolver is improved for models with many MPC-type constraints.

Benefits: This enhancement reduces analysis run times significantly for common flexible body generation workflows.

The AMS full recovery performance is improved significantly for models with many MPC-type constraints by minimizing the SIM I/O operations. The performance improvement can be frequently demonstrated for a large substructure generation model with many retained nodes and a nondefault flexible body generation request that imposes the AMS full recovery. This improvement will be especially significant for the generation of large flexible bodies for particular multibody dynamics solvers with many retained nodes.

For example, the figure below illustrates the improved performance of Abaqus/Standard using the AMS eigensolver for two benchmark models run on a machine with Intel Xeon Gold 6248 (2.5 GHz Cascade Lake processor with 2 × 20 cores) and 1.5 TB memory.



The first model is a heavy-duty engine model with 15 million degrees of freedom (DOFs) extracting 8 eigenmodes and 5706 substructure modes, and the second model is a trim body model with 26 million DOFs extracting 2547 eigenmodes and 2831 substructure modes. The figure shows how the AMS full recovery workflows have been improved incrementally in 2024 FD03 and 2025 FD01.

The first model took 3.5 hours to run in 2024 GA. This run time was reduced to 2.8 hours in 2024 FD03, and it is further reduced to 1.7 hours in 2025 FD01 due to the improved MPC recovery workflow. As a result, the job in 2025 FD01 runs twice as fast as the one in 2024 GA. Similarly, the job for a 26M DOF trim body model in 2025 FD01 runs 1.4 times faster than in 2024 GA.

For more information, see *Natural Frequency Extraction*.

Additive Manufacturing: Scan Pattern Assignment Using an Element Set

In an Abaqus/Standard additive manufacturing process simulation employing the scan pattern approach, you can now assign a scan pattern directly to the elements contained in an element set.

Benefits: The enhancement facilitates the assignment of scan patterns in situations when different regions in the model cannot be easily separated by a box definition.

You can combine scan pattern assignments using the new element set–based approach with the preexisting box-based assignments. In addition, you can use the new approach for both eigenstrain-based and thermomechanical simulations of additive manufacturing processes.

For more information, see *Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Pattern-Based Method* and *Eigenstrain-Based Analysis of Additive Manufacturing Processes*.

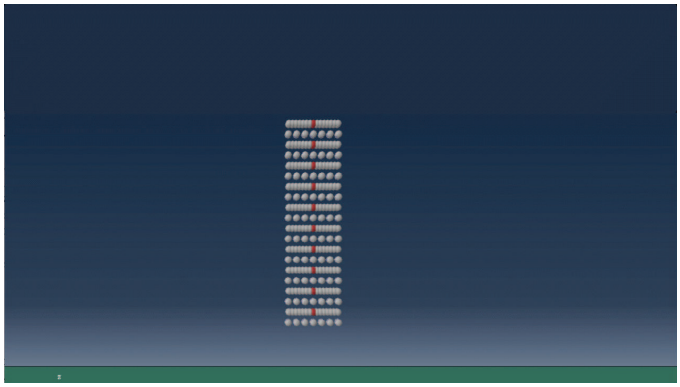
Particle Clustering for Discrete Element Analysis

You can now define arbitrary grain geometry in an Abaqus/Explicit discrete particle analysis by clustering particles together.

Benefits: The new capability allows you to define more realistic nonspherical grain geometry, which improves the fidelity of a discrete element method (DEM) analysis.

Abaqus/Explicit now supports the definition of arbitrary grain geometries for discrete particle analysis by allowing you to define clusters of particles that are rigidly held together to represent the geometry of the physical grains. Each cluster has one parent particle and an arbitrary number of child particles. The particles in a cluster might or might not overlap with each other. The cluster moves as a rigid body with its reference point at the center of the parent particle. (The cluster has three translational and three rotational degrees of freedom at the reference node.) The new DEM particle cluster functionality supports all existing contact interactions.

Only the DMP execution mode is supported for the new DEM particle cluster functionality. The animation below shows a DEM model of a beam stack under gravity loading, with individual beams modeled as clusters of particles. The parent particles have been color-coded in red and blue, while child particles are shown in gray.



For more information, see *Discrete Element Method* and *Particle cluster drop*.

Flexible Body Generation Performance

Flexible body generation performance is improved substantially.

Benefits: You can complete the flexible body generation analysis faster than in earlier releases.

Abaqus/Standard can generate a flexible body from a substructure. It supports several flexible body types for several external flexible body dynamics solvers. Generating the flexible body entities involves massive calculations and significantly affects the analysis time.

The enhancements improve performance for analyses generating flexible bodies for the Simpack multibody dynamics solver, for the AVL EXCITE™ flexible body dynamics solver from AVL LIST GmbH, and for the ADAMS™ flexible body dynamics solver from MSC.Software Corporation.

To demonstrate the performance improvements, two benchmark models are run on Intel Xeon Gold 6348 (2.6 GHz Ice Lake processor with 56 physical cores) with 2 NUMA nodes and 2 sockets.

Model 1. Car body-in-trim structure with 26 million finite element degrees of freedom, 384 substructure retained degrees of freedom (static modes), and 1130 substructure dynamic modes.

Model 2. Engine structure with 12 million finite element degrees of freedom, 2028 substructure retained degrees of freedom (static modes), and 1400 substructure dynamic modes.

The table below illustrates the improved performance of the flexible body generation.

Benchmark model	Flexible body type								
	ADAMS™			AVL EXCITE™			Simpack		
	R2024	R2025	Gain	R2024	R2025	Gain	R2024	R2025	Gain
Model 1									
Flex. Body Gen. time, sec	68950	1830	**37.7×**	7611	785	**9.7×**	14714	1645	**8.9×**
Abaqus/Standard time, sec	72258	5218	**13.8×**	10962	4252	**2.6×**	18053	4995	**3.6×**
Disk peak usage, GB	418	420	**~1×**	762	418	**1.8×**	763	418	**1.8×**

Benchmark model	Flexible body type								
	ADAMS™			AVL EXCITE™			Simpack		
	R2024	R2025	Gain	R2024	R2025	Gain	R2024	R2025	Gain
Model 2									
Flex. Body Gen. time, sec	14886	3931	**3.8×**	21779	1230	**17.7×**	50468	2789	**18.1×**
Abaqus/Standard time, sec	16382	5544	**3.0×**	23298	2774	**8.4×**	51975	4374	**11.9×**
Disk peak usage, GB	372	369	**~1×**	990	370	**2.7×**	990	369	**2.7×**

For more information, see *Generating Substructures*.

PDF Guides Available

You can download PDF versions of the Abaqus guides from each guide overview.

Benefits: You can easily access the PDF versions of the guides.
In earlier releases, PDF versions of the guides were updated only for the GA (General Availability) release and could be downloaded from the Dassault Systèmes Knowledge Base. As of 2025 FD01, the PDF guides will be updated with each release of the HTML versions of the guides.

For more information, see *Abaqus Analysis Guide*.

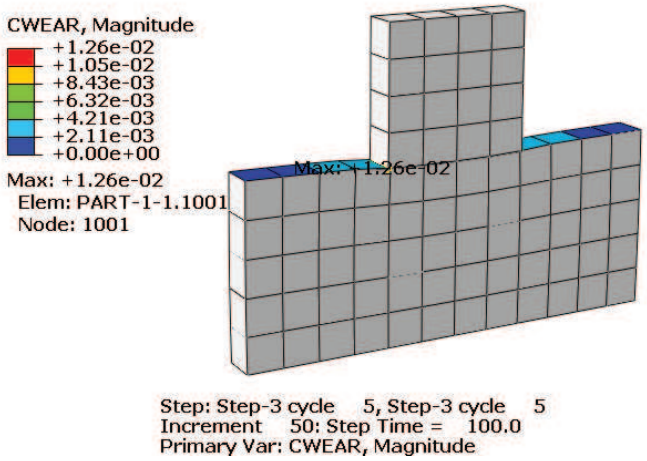
2025 GA

Step Cycling

The step cycling capability provides a simple way to repeat a step or a series of steps over a prescribed number of cycles.

Benefits: You can define cyclic repetition of a sequence of steps using the new step cycling capability instead of explicitly defining steps for every cycle, which provides simpler model setup, better performance, and the ability to end step cycling based on the analysis solution.
There are situations where the analysis requires the repetition of a step, or a series of steps, over a specified number of cycles. Examples of such workflows include aging in batteries, brake wear analysis, and fatigue. In earlier releases, the only approach to repeat a step, or a series of steps, in an analysis was through repeated definition of the same step or steps multiple times. This resulted in reduced performance during all stages of a model (preprocessing, analysis, and output).

The use of step cycling provides an efficient mechanism to loop over the sequence of steps with significant performance improvements. It also provides mechanisms to end a step sequence prematurely based on certain user-specified criteria. A step cycle sequence can be preceded or followed by steps and/or another cycle sequence.



The animation above shows wear distance accumulation on the slider and the bottom block as the slider repeatedly moves back and forth. In this example, the sample wear cycle is modeled using two steps. In “Step-2” the slider moves to the right and returns to the center, while in “Step-3” the slider moves to the left and returns to the center. You can find additional details on this example in *Step Cycling for Contact Wear Simulations*.

For more information, see *Step Cycling*, *Modeling Aging in Batteries*, and *Contact Wear*.

2024 FD03

Alternative Convergence Checks

A new set of alternative convergence controls are now available in Abaqus/Standard.

Benefits: In simulations that involve strong multiphysics coupling among a number of different fields, the alternative convergence controls, which are typically more relaxed compared to the default convergence controls, can result in better convergence characteristics and faster solution times with minimal effects on accuracy.

While the traditional convergence checks in Abaqus/Standard work well for structural simulations, they might be too restrictive for some multiphysics problems (such as battery electrochemistry simulations) that involve strong coupling among a number of different physical fields.

In earlier releases, the only way to modify the default convergence behavior in Abaqus/Standard was to specify nondefault tolerances on the maximum residuals and corrections during an increment. The new alternative convergence checks provide greater flexibility and can be used either globally (for all fields that are active in the analysis) or on an individual field basis. You can also modify the choice of the convergence checks at the step level.

Important: The use of this set of features requires some experience and testing to ensure that the alternative convergence measures produce results with an acceptable level of accuracy.

The table below shows a comparison of solver passes based on different convergence checks.

Model	Number of DOFs	Iteration Count		Gain
		Convergence Check = Strict	Convergence Check = Relaxed	
Model 1	792	1998	1662	1.20
Model 2	2232	673	357	1.88
Model 3	82692	156	128	1.22
Model 4	2380141	1417	958	1.48
Model 5	2764054	1998	1662	1.20
Model 6	4457894	545	395	1.38

For more information, see *Modifying the Default Convergence Checks*.

Adjoint Design Sensitivity Analysis

The adjoint design sensitivity functionality is enhanced significantly for eigenfrequency and direct steady-state dynamic analyses.

Benefits: You can perform the adjoint design sensitivity analysis faster, more capabilities are available, and the usability of the adjoint design sensitivity analysis is improved.
The following enhancements are available:

- Eigenfrequency adjoint design sensitivity analysis now supports the AMS eigensolver. In earlier releases, you could use only the Lanczos eigensolver for this analysis type. Now, you can run adjoint design sensitivity analyses for larger models and take advantage of parallel scaling in the AMS eigensolver.
- Direct steady-state dynamic adjoint design sensitivity analysis now supports topology and shape optimization. In earlier releases, you could use only sizing optimization for this analysis type.
- Direct steady-state dynamic adjoint design sensitivity analysis and eigenfrequency adjoint design sensitivity analysis now allow you to specify sizing and topology design variables for the same element.

For more information, see *Adjoint Design Sensitivity Analysis Overview*.

State Space Solver for Transient Modal Dynamic Analysis

You can use the new state space solver to perform transient modal dynamic analysis for models with nondiagonal modal damping operators.

Benefits: The new solver is unconditionally stable and offers improved performance for models with many time increments and a moderate number of modes. It also supports coupled structural-acoustic analysis.

In earlier releases, the only solver available for transient modal dynamic analysis with nondiagonal modal damping was the conventional solver. The conventional solver uses a time integration scheme based on the assumption that the nondiagonal entries of the modal damping operator vary linearly within each time increment together with the loads. It calculates the approximate modal solution, and the solution accuracy depends on the time step size. In addition, the time integration algorithm is not unconditionally stable, particularly for models with a high level of damping. For common engineering simulations, the accuracy and stability of the conventional solver is usually acceptable. However, obtaining an accurate and stable solution using the conventional solver

requires the use of very small time increments for specific cases. In addition, the conventional solver does not support coupled structural-acoustic analysis.

The new state space transient modal dynamic solver calculates the exact modal solution for systems with arbitrary modal damping for loads that vary piecewise linearly with time. This solver uses the complex eigenmodes of the unsymmetric modal state space operator to represent the real transient modal solution. The solution includes two phases: solving the unsymmetric eigenvalue problem for the state space operator at the "setup" phase and calculating the transient solution at the "apply" phase. The setup phase is the most computationally intensive part of the analysis. It can be costly for systems with many thousands of modes. However, the calculated solution with the state space solver is more accurate than for the conventional solver and the algorithm is unconditionally stable. In addition, the state space solver supports structural-acoustic transient modal analysis using the modal subspace built from the coupled structural-acoustic eigenmodes or from the uncoupled structural and acoustic eigenmodes.

For more information, see *Transient Modal Dynamic Analysis*.

Unit System Translation

The SIMULIA Co-Simulation Services now supports unit system translation.

Benefits: You now have the flexibility to use the unit system best suited to your industrial workflow.

To avoid mistakes, it is best to perform coupled multiphysics and multiscale simulation where both models use the same units. Abaqus has no units built into it and assumes self-consistent units. Some third-party solvers might represent their internal data in a unit system different from the one in which the user defined the model.

The SIMULIA Co-Simulation Services typically tabs into the solver and consequently exchanges field data in the solver's unit system. This forces the Abaqus model to be in the unit system used by the other solver's internal format and can lead to unconventional model specification. For example, consider a sequential analysis of a circuit board modeled in CST Studio Suite with results imported into Abaqus. You might employ the CGS system (centimeter-gram-seconds) to define the electromagnetic model. However, CST Studio Suite converts everything to SI (International System) units to perform calculations. This forces the Abaqus model to be in SI units and to use meters as the length dimension for model definition (geometry, material data, load, etc.), which is an unconventional choice for this particular workflow.

Translation between unit systems is available only if the following are true:

- the unit system is defined in the Abaqus model and
- the other solver supports unit system translation through the co-simulation service.

For more information, see *Defining a Unit System for Translating Data with Other Solvers* and **UNIT SYSTEM*.

DMP Direct Sparse Solver Enhancements

The backward pass phase in the distributed memory parallel (DMP) direct sparse solver is improved.

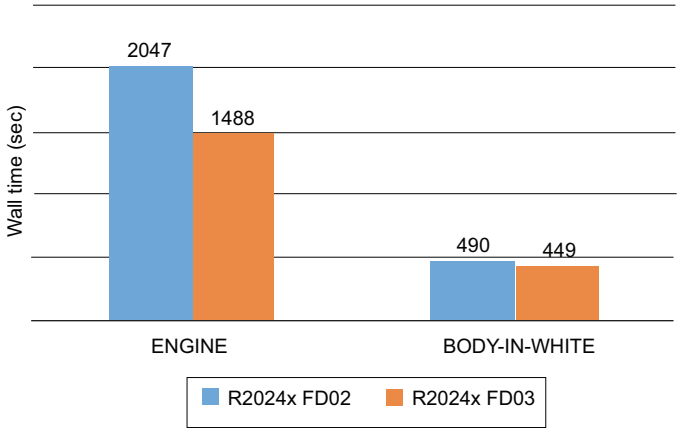
Benefits: The enhancements to the DMP direct sparse solver allow you to complete many analysis procedures faster.

Performance improvements are expected in workflows dominated by the backward pass; for example,

- static substructure generation using the conventional (not AMS-based) algorithm,
- linear perturbation static or steady-state dynamic analysis with multiple load cases, and
- eigenvalue extraction using the Lanczos eigensolver.

The improvements also eliminate cases where the solver could stop responding in certain DMP configurations. While this enhancement improves performance of the backward pass when the factorized matrix resides entirely in system memory, it might degrade performance for cases where the factorized matrix must be partially stored on disk.

To demonstrate performance improvements, two benchmark models are run on Intel Xeon Gold 6348 (2.6 GHz Ice Lake processor with 56 physical cores), with 2 NUMA nodes, and 2 sockets.



- **Model 1:** Static substructure generation for an engine model with 12M DOFs with 2028 retained DOFs run using a host configuration of 1 × 1 × 56 (1 node, 1 mpi rank, 56 threads). This model shows 1.4x speedup in 2024 FD03 compared to 2024 FD02.
- **Model 2:** Direct steady-state dynamic analysis for a body-in-white car model with 3.6M DOFs with 20 frequency points and 120 load cases run using a host configuration of 2 × 8 × 7 (2 nodes, 8 mpi ranks, 7 threads). This model shows 1.1x speedup in 2024 FD03 compared to 2024 FD02.

For more information, see *Direct Linear Equation Solver*.

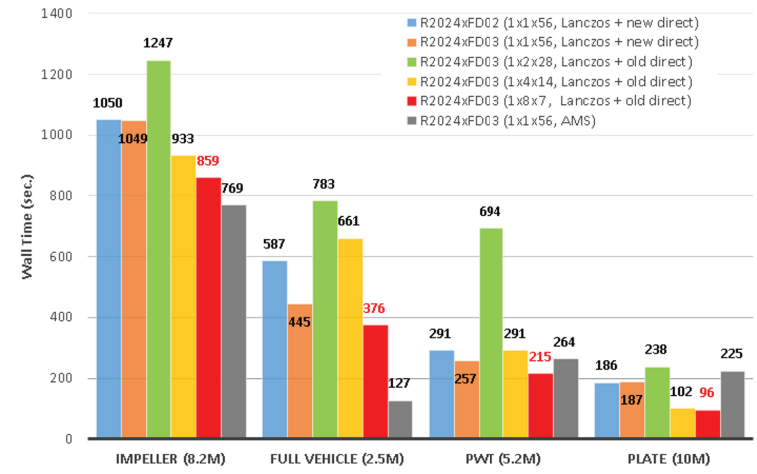
MPI-Based Parallel Execution of the Lanczos Eigensolver

MPI-based parallel execution of the Lanczos eigensolver is now available.

Benefits: Frequency extraction analysis using the Lanczos eigensolver in hybrid mode, which leverages both MPI and thread parallelism, runs significantly faster and can solve much larger problems by utilizing multiple compute nodes.

You can now run frequency extraction analyses using the Lanczos eigensolver in a hybrid mode that utilizes both MPI and thread parallelism. By leveraging both MPI and thread parallelism within a single compute node, as well as across multiple compute nodes on the HPC cluster, the new Lanczos eigensolver significantly enhances the performance of Lanczos eigensolver workflows.

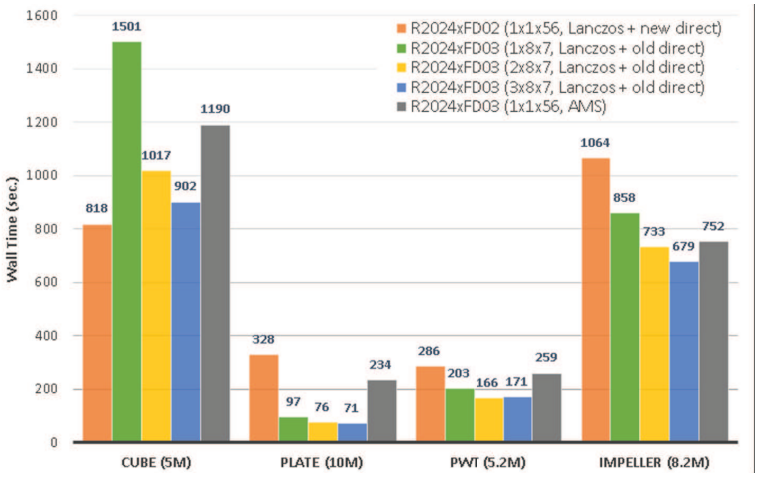
The figure below shows the performance improvement of Abaqus/Standard using the new Lanczos eigensolver in 2024 FD03 for four benchmark models. Each column represents an Abaqus/Standard run time in a different host configuration. Because of the new enhanced thread-based direct solver released in 2024 GA, the Lanczos eigensolver uses the new enhanced thread-based direct sparse solver in shared memory parallel (SMP) mode and the DMP direct sparse solver available in earlier releases in hybrid mode.



The figure illustrates the following:

- The Lanczos eigensolver in SMP mode in 2024 FD03 shows better performance than the previous version in 2024 FD02 for some of models due to the enhancements in the eigenmode transfer phase.
- The Lanczos eigensolver in hybrid mode in 2024 FD03 shows good parallel scaling using multiple MPI ranks on a single compute node. It also shows significantly improved performance compared to the eigensolver in 2024 FD02. For the full vehicle model with 2.5M DOFs, for example, Abaqus/Standard using a host configuration of 1 x 8 x 7 in 2024 FD03 shows 1.6 times speedup compared to 2024 FD02.
- For all four benchmark models, the Lanczos eigensolver in hybrid mode (1 x 8 x 7) runs faster than the Lanczos eigensolver in SMP mode. It also runs faster than the AMS eigensolver with full recovery for some models in 2024 FD03. Therefore, the new Lanczos eigensolver in hybrid mode surpasses the performance limitations of the eigensolvers in SMP mode.

The figure below shows the performance improvement of the new Lanczos eigensolver when utilizing multiple compute nodes on the HPC cluster.



The figure illustrates the following:

- The new Lanczos eigensolver shows good parallel scaling using multiple compute nodes.

- The new Lanczos eigensolver using multiple compute nodes (that is, using more CPU cores and memory) runs significantly faster than the Lanczos eigensolver and the AMS eigensolver in SMP mode for all four models.
- For the impeller model with 8.2M DOFs, for example, Abaqus/Standard using a host configuration of 3 x 8 x 7 in 2024 FD03 runs 1.6 times faster than the previous version in 2024 FD02. It is even faster than the AMS eigensolver in 2024 FD03. This demonstrates that the new Lanczos eigensolver in hybrid mode using multiple compute nodes surpasses the performance limitations of the eigensolvers in SMP mode.

For more information, see *Parallel Execution in Abaqus/Standard*.

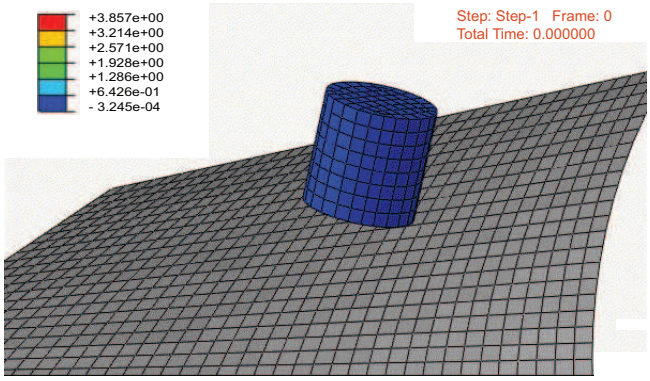
Parallel Execution for SPH Progressive Element Conversion

You can now run SPH progressive element conversion models in parallel.

Benefits: Parallel execution support can greatly improve the simulation performance of models that include SPH progressive element conversion.

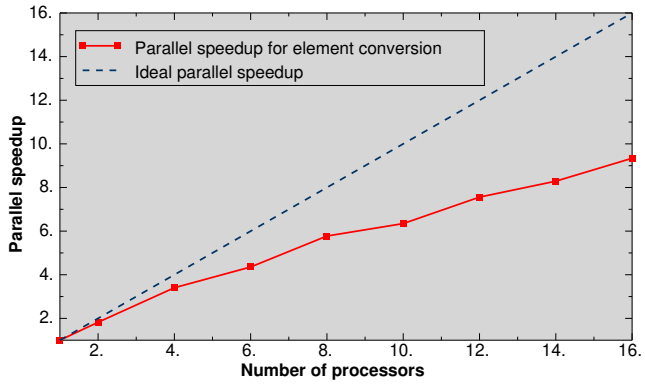
Earlier releases of Abaqus/Explicit supported SPH progressive element conversion only in serial execution mode, limiting the usability of this feature for broad applications. This release supports parallel execution for SPH progressive element conversion, which enables you to take full advantage of the intrinsic strengths of both the Lagrangian finite element and SPH methods.

The figure below shows a parallel run example for a model of a bird striking an airplane engine blade.



The bird structure consists of 480 C3D8R elements, with an EOS material definition that can be converted to around 100k SPH particles based on a strain conversion criterion. Once the strain for a certain C3D8R element meets the user-defined conversion criterion, the element is converted to SPH particles to sustain more severe deformation.

The graphs below show the parallel execution speedup as function of the number of processors, illustrating reasonably good scaling.



The parallel speedup might vary for different models depending on the uniformity of the SPH particle distributions. It is recommended to use more uniform-sized parent elements (preferably C3D8R) for the SPH progressive element conversion to achieve better accuracy and parallel scalability.

Fatigue Crack Growth Analysis Using Field Import-Based Submodeling

You can now import external fields in a linear elastic fatigue crack growth analysis.

Benefits: You can now predict fatigue crack growth in submodels through external field import by mapping fields from the global model to the boundary of the submodel. This allows you to use global models with multiple steps.
In the traditional submodeling approach, the submodel can be driven only by results from a single step in a global model.

For more information, see *General Capability for Importing External Fields*, *Linear Elastic Fatigue Crack Growth Analysis*, and *Node-Based Submodeling*.

2024 FD02

Element Design Responses for Transient Adjoint Design Sensitivity Analysis

Transient sensitivity analysis now supports element design responses based on stress and plastic strain as well as user-defined responses.

Benefits: Element design responses, such as stresses and various measures of plastic strain, allow design optimization for transient dynamic workflows to set constraints on the maximum von Mises stress and/or various measures of the plastic strain.
In earlier releases, Abaqus limited the list of available design responses in a transient sensitivity analysis primarily to nodal responses. The rate of internal work was the only available element-level response. The list of element design responses is now expanded significantly to include different stress and/or plastic strain measures.

In addition, you can use user subroutine *UELEMDRESP* to define a user-defined element design response, such as a failure criterion, in a transient optimization.

For more information, see *Adjoint Design Sensitivity Analysis Overview* and *UELEMDRESP*.

Specifying Design Responses in an Adjoint Design Sensitivity Analysis

You can now specify design responses in an adjoint design sensitivity analysis. In addition, you can prevent sensitivity calculations and output that might not be important for design optimization.

Benefits: You can manage the specification of design responses more effectively and prevent the output database from becoming too large.

The list of available operators for node and element design responses now includes the maximum absolute and the minimum absolute operators. These operators can be helpful in situations such as the following:

- In the context of node responses, the maximum absolute operator is useful when the nodal displacements for a node set that contains many nodes should be constrained to be less than a certain value. Without the maximum absolute operator, you must define one node response for each node in the node set, leading to many node responses (and corresponding adjoint sensitivities) in the output database. When you use the maximum absolute operator, you need to define only a single design response that uses this design objective or constraint.
- The minimum absolute operator is useful when the reaction force for a node set that contains many nodes must be larger than a certain value. Without the minimum absolute operator, you must define one node response for each node in the node set, leading to many node responses (and corresponding adjoint sensitivities) in the output database. When you use the minimum absolute operator, you need to define only a single design response that uses this design objective or constraint.
- You can now specify lower and upper bounds on the values of a design response outside of which Abaqus will not compute the adjoint sensitivities. By default, Abaqus writes the design response and the corresponding adjoint sensitivities to the output database for all design responses that do not have any prescribed bounds for the design response definition. In some situations, the sensitivities are necessary only if the value of a design response is within a certain range. In these cases, specifying the lower and upper bounds on the design response values provides a convenient way to limit the sensitivity calculations and the size of the output database.

For more information, see *Adjoint Design Sensitivity Analysis Overview*.

Structure-to-Structure Co-Simulation Using Abaqus/Standard and Abaqus/Explicit

The keyword interface has changed for supporting structure-to-structure co-simulation using Abaqus/Standard and Abaqus/Explicit.

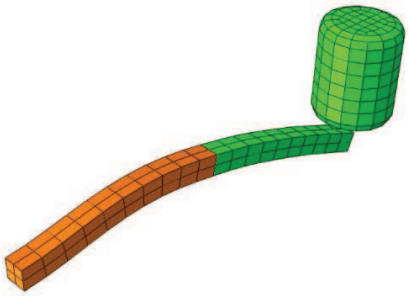
Benefits: The new keyword interface is aligned with other co-simulation products and supports improved interface computations that improve structure-to-structure co-simulations.

Co-simulation between Abaqus/Standard and Abaqus/Explicit exhibits strong physics coupling and requires special treatment at the co-simulation interface. The enhanced subcycling method (see *Enhanced Subcycling Method for Interface Calculations*), which improves solution robustness and is more computationally efficient, was introduced in an earlier release. The enhanced subcycling method is recommended over the original subcycling method (see *Original Subcycling Method for Interface Calculations*) and the lockstep method (see *Lockstep Method for Interface Calculations*).

The original subcycling method and the lockstep method used the **CO-SIMULATION*, PROGRAM=ABAQUS option. Since these are no longer the recommended interface treatments, the PROGRAM=ABAQUS parameter is being deprecated. In addition, the *CO-SIMULATION CONTROLS option, which selects between the original subcycling and lockstep algorithms, is also being deprecated.

You can still select any of the above interface treatment methods by specifying the fields exchanged explicitly and by using the appropriate co-simulation service configuration file, as described in *Structural-to-Structural Co-Simulation*.

The image below shows a cantilever beam impacted by an object. The beam is split into two domains: the orange portion is modeled in Abaqus/Standard, and the green portion is modeled in Abaqus/Explicit.



For more information, see *Structural-to-Structural Co-Simulation*.

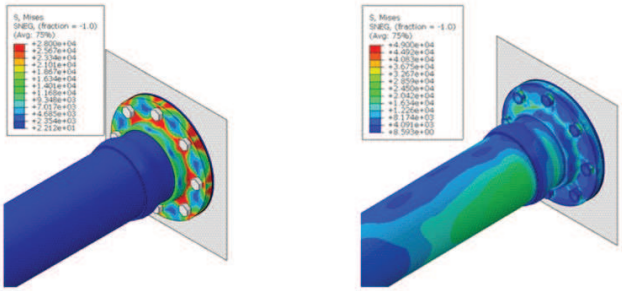
Full-Model Input for Abaqus Import

You can use the full-model input format to specify both new and imported model data in an Abaqus/Standard or Abaqus/Explicit import analysis.

Benefits: The full-model input format allows flexibility in import feature management; for example, partial import of element sets of a section, import of previously rigid element sets as deformable, and import analysis using the SIM results format.

When you use the full-model input format, you give imported model data (such as nodes and elements, node sets and element sets, and sections and materials) in the input. This provides complete model information used in the import analysis.

By using the full-model input format, you can perform certain import workflows (such as importing a previously rigid element set as deformable) simply by omitting the rigid body definition of the element set in the import input. The following example illustrates this approach. The bolts are defined as rigid in the first analysis (left), and they are imported as deformable in the import analysis (right).



For more information, see *About Transferring Results between Abaqus Analyses*.

Undrained Pore Fluid Flow and Stress Analysis

You can now perform an undrained pore fluid flow and stress analysis in Abaqus/Explicit.

Benefits: You can account for the presence of a permeating pore fluid and its effect on the mechanical response of a saturated granular solid skeleton under dynamic loading conditions.

A porous medium typically consists of a solid skeleton of numerous solid grains that are “loosely packed” and a permeating pore fluid that occupies a system of interconnected pores among these solid grains. You can ignore the effect of fluid flow in situations when the loading rate is significantly faster than the rate at which pore fluid can drain from the porous medium. This corresponds to the undrained conditions.

The undrained pore fluid flow and stress analysis functionality in Abaqus/Explicit is intended for the simulation of fully saturated porous media under undrained conditions. The functionality enables modeling the increase in stiffness of the soil in applications such as trenching, gouging, and pile driving. In these applications, you can ignore fluid diffusion terms when the rate of loading is fast compared to the characteristic time of the diffusion process.

For more information, see *Undrained pore fluid flow and stress analysis*.

2024 FD01

Iterative Solver Performance Improvements

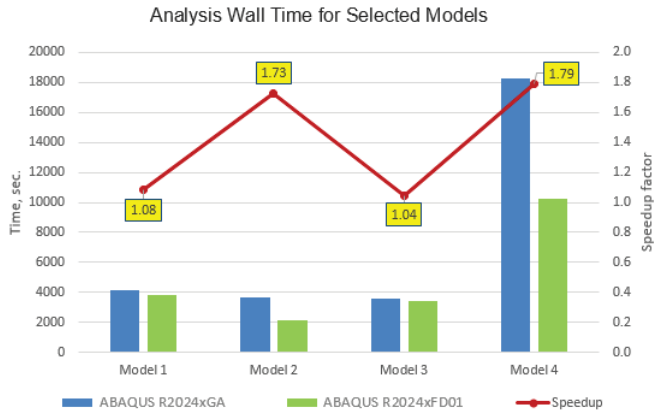
A new domain decomposition algorithm improves iterative solver convergence rates and shortens the overall solution time.

Benefits: The iterative solver is now able to converge on models on which it could not converge in earlier releases (for example, automotive battery structural simulations) or on models for which convergence was prohibitively slow (for example, electric engine analyses).

The new domain decomposition algorithm takes into account the connectivity of the finite element model, the element formulation details, and the material properties. It creates a better clustering of elements with similar properties and separates out clusters with dissimilar properties. This makes it easier for the iterative solver to find a smooth solution in individual aggregates, which results in improved convergence rates overall. The algorithm optimizes the load balance between domains for large models in parallel runs by accounting for the actual computational costs of different elements. The solution time reduction due to the improved domain decomposition varies from model to model. The improvement can be moderate for some finite element models, but it can be significant for large-scale models with complicated geometry using higher-order element formulations.

The figure below illustrates the improved performance of the iterative solver for four selected finite element models:

- Model 1: airplane fuselage structure; shell-dominated finite element model with 48.3 million degrees of freedom.
- Model 2: powertrain model with 23.6 million degrees of freedom meshed with linear continuum elements.
- Model 3: powertrain model with 11.2 million degrees of freedom meshed with quadratic continuum elements.
- Model 4: battery model with 46.0 million degrees of freedom, meshed with linear continuum, shell, and connector elements.



For more information, see *Iterative Linear Equation Solver*.

Generating Reduced Bases for Use with Nonlinear Flexible Multi-Body Systems

A new analysis technique in Abaqus/Standard reads operator data from upstream analyses and generates reduced bases for use in downstream nonlinear flexible multi-body system (MBS) simulations with Simpack.

Benefits: This streamlines the process for preparing MBS simulations that capture large deformation effects with high accuracy and fast run times.

A reduced basis is a set of basis vectors that can represent solutions from preceding analysis cases when they are superimposed with according scaling factors. You can now generate reduced bases for use in downstream nonlinear flexible MBS simulations with Simpack as follows:

1. Define a nonlinear static analysis case that ramps up a large deformation of a flexible component in a shape that is characteristic for its behavior in downstream system simulations. For example, apply a distributed load that acts on the surface of a wind turbine rotor blade in the flap-wise direction, or define a spring support for a twist beam axle and apply a twisting load at the wheel attachment points.
2. Output the operator data at various deformation levels.
3. Add natural frequency extraction steps at various deformation levels.
4. Add the reduced basis generation step.

The reduced basis generation step computes the reduced basis from the solution data and computes the auxiliary operator data from all the preceding analysis steps. The reduced basis and projected operator data are written to a SIM (`.sim`) file. Optionally, you can write a Simpack Flexible Body Interface (`.fbi`) file with a complete Simpack nonlinear flexible body definition that you can assign directly to a nonlinear flexible body in Simpack.

For more information, see *Reduced Basis Generation* and **REDUCED BASIS GENERATE*.

Random Response Analysis Performance Improvements

Random response analyses now have improved performance for the element results computation and output.

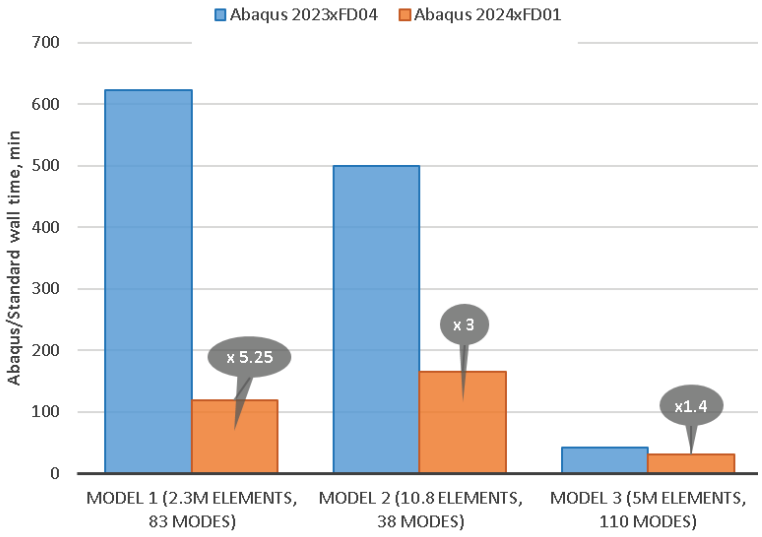
Benefits: The improvements reduce the Abaqus/Standard analysis wall time for random response analyses, particularly for large models with extensive element field output requests.

The enhancement substantially improves performance of the following phases of the random response analysis:

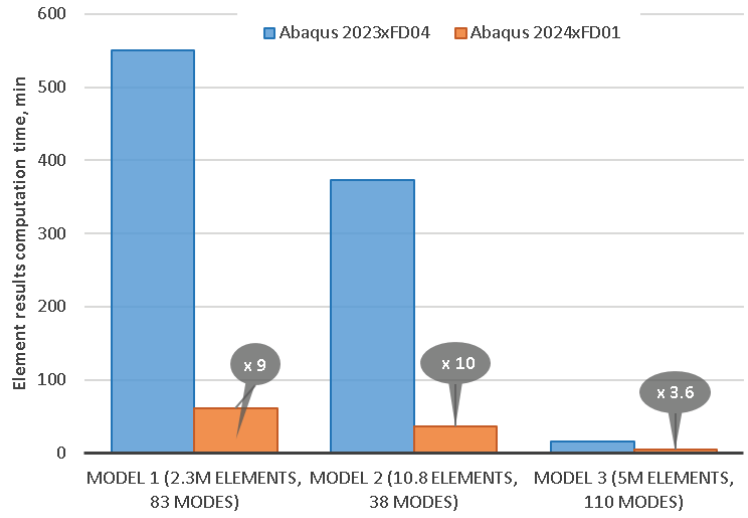
- Reading the eigenfrequency extraction stresses.
- Computing the element results. New scalable algorithms perform the element output computations.

In addition, you can now use the frequency split technique to run a random response analysis on models that do not require calculations of the root-mean-square variables.

The figure below illustrates the wall time improvements for running a random response analysis in Abaqus/Standard for three industrial models.



The figure below illustrates the reduction in the wall time for the element results computation phase for the same three models.



The models and results are as follows:

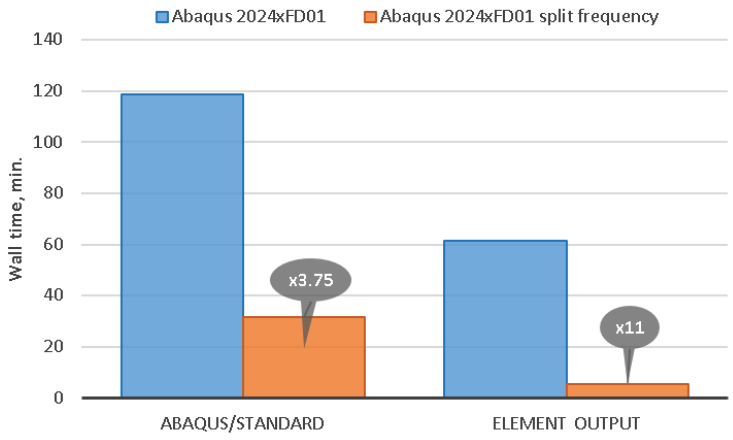
- Model 1 is comprised of 2.3 million elements containing approximately 9 million degrees of freedom. The random response analysis was performed using 83 eigenmodes and 840 frequency points. Stress field output was requested in all increments.

  The Abaqus/Standard wall time was reduced approximately 5.25 times, while the element results computation phase was sped up approximately 9 times.
- Model 2 is comprised of 10.8 million elements containing approximately 53 million degrees of freedom. The random response analysis was performed using 38 eigenmodes and 20 frequency points. Stress and strain field output was requested in all increments.

  The Abaqus/Standard wall time was reduced approximately 3 times, while the element results computation phase was sped up approximately 10 times.
- Model 3 is comprised of 5 million elements containing approximately 25 million degrees of freedom. The random response analysis was performed using 110 eigenmodes and 38 frequency points. Stress and Mises stress field output was requested in 0.6 million elements at all frequency points.

  The Abaqus/Standard wall time was reduced approximately 1.4 times, while the element results computation phase was sped up approximately 3.6 times.

The figure below shows the result of applying the frequency split technique to Model 1. The 840 frequency points used for the random response analysis were split into 16 batches. The total Abaqus/Standard wall time (that includes the wall time of a frequency extraction step and the largest wall time of the random response analyses) was reduced approximately 3.75 times, while the element results computation phase was reduced approximately 11 times.



For more information, see *Random Response Analysis*.

AMS Eigensolver Performance Improvement

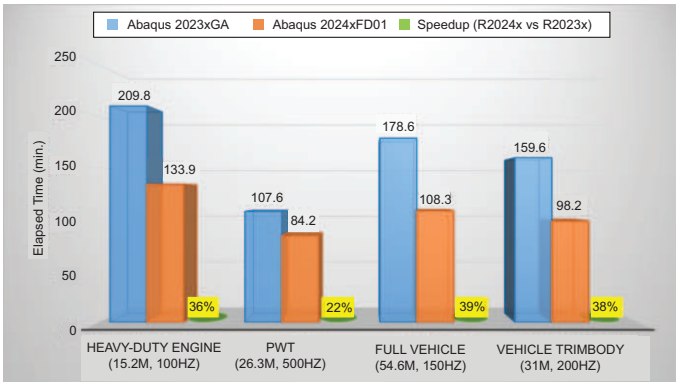
The performance of the frequency extraction analysis using the AMS eigensolver has been improved for models that require full eigenmodes and substructure modes.

Benefits: The AMS full recovery performance improvement significantly reduces the analysis run times for an AMS frequency extraction analysis and for an AMS-based substructure generation analysis requesting full eigenmodes.

The performance of the following phases related to full eigenmodes recovery is significantly improved for the AMS eigensolver workflow including the AMS-based substructure generation procedure:

- AMS recovery phase
- Constraint transformation and recovery phases
- Substructure recovery matrix output phase

The figure below illustrates the performance improvement of Abaqus/Standard in Abaqus2024 FD01 for four industrial models. In this figure, the blue columns represent the Abaqus2023 GA run time, and the orange columns represent the Abaqus2024 FD01 run time.



The models and results are as follows:

- Model 1 is a heavy-duty engine model with 15.2 million degrees of freedom, extracting 10 eigenmodes below 100Hz and 5,704 substructure modes for 5,704 retained degrees of freedom.

  Abaqus2024 FD01 shows a 36% performance improvement over Abaqus2023 GA.
- Model 2 is a powertrain model with 26.3 million degrees of freedom, extracting 6 eigenmodes below 500Hz and 1,512 substructure modes.

  Abaqus2024 FD01 shows a 22% performance improvement over Abaqus2023 GA.
- Model 3 is a full vehicle model with 54.6 million degrees of freedom, extracting 1,630 eigenmodes below 150Hz and 1,984 substructure modes.

  Abaqus2024 FD01 shows a 39% performance improvement over Abaqus2023 GA.
- Model 4 is a vehicle trim body model with 31 million degrees of freedom, extracting 2,598 eigenmodes below 200Hz.

  Abaqus2024 FD01 shows a 38% performance improvement over Abaqus2023 GA.

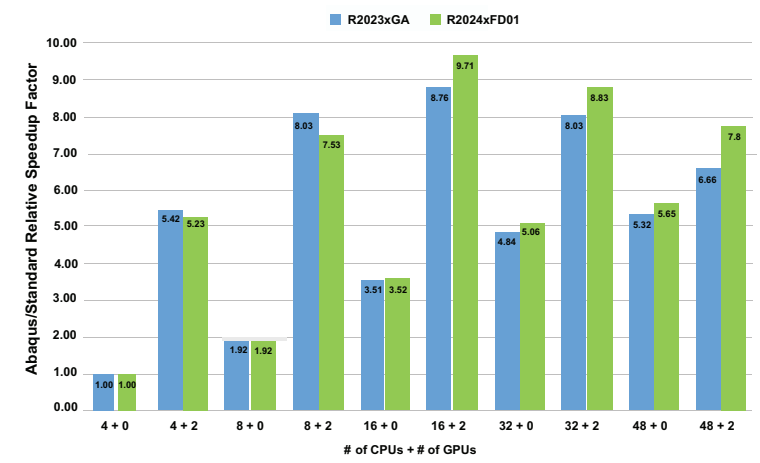
For more information, see *Natural Frequency Extraction*.

AMS Eigensolver GPU Acceleration

Robustness and performance of the AMS eigensolver GPU acceleration is improved for frequency extraction analyses with new NVIDIA GPU cards. New GPU acceleration functionality speeds up modal damping projection in the AMS recovery phase.

Benefits: Analyses using the AMS eigensolver run faster using multiple GPUs.

The figure below shows the GPU acceleration performance of Abaqus/Standard for a 7.6 million degree of freedom stator model that requires 2.3×10^{15} floating-point operations to factorize four system operators (including viscous and structural damping matrices).



The blue columns represent the Abaqus2023 GA run time, and the green columns represent the Abaqus2024 FD01 run time. The performance data was obtained on a machine equipped with Intel Xeon Gold 6336Y (Ice Lake, 2.4GHz, 48 cores) processors, 512 GB physical memory, and 2x NVIDIA A100 (40 GB memory) cards.

Abaqus/Standard using the AMS eigensolver shows good parallel scaling up to 48 CPUs. However, using 16 CPUs and 2 GPUs results in the best performance of the GPU acceleration for this model.

For more information, see *Parallel Execution in Abaqus/Standard*.

Scalable Spectral Solver for Mode-Based Steady-State Dynamics Analysis

A new scalable spectral solver is available for common cases of mode-based steady-state dynamics analysis.

Benefits: Using the new solver can significantly reduce the analysis time for damped finite element models with general structural damping and specific viscous damping properties.

You can use the new spectral solver for the special cases of mode-based steady-state dynamic simulations. For all cases, you can apply any kind of structural damping in the model. The special cases are defined by the complementary viscous damping properties.

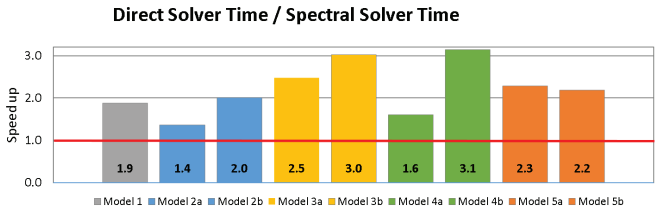
Case 1	No viscous damping	No material, global, or modal viscous damping
		No dashpot elements, connector damping, or substructures with viscous damping
		No matrix input data for the viscous damping

		No other possible sources of the viscous damping
Case 2	Low rank viscous damping operator	Viscous damping is defined locally in the model, for example, using dashpot elements, connector damping, or substructures with viscous damping
		Structural-acoustic coupled steady-state dynamic analysis using uncoupled eigenmodes when the number of acoustic modes is significantly smaller than the number of the structural modes
		Modal viscous damping defined for selected modes

You can use the spectral solver for mechanical, acoustic, and structural-acoustic analyses. Using the spectral solver can be beneficial for models with the mentioned special damping properties when many eigenmodes and many points in the frequency domain are used for the frequency response calculation.

The spectral solver algorithm consists of two analysis phases: the Setup Solver Phase and the Apply Solver Phase. The computational cost of the Setup Solver Phase depends on the number of the eigenmodes, and it can be significant when that number is high. The computational cost of the Apply Solver Phase is proportional to the number of the frequency points specified for the frequency response calculation, and the solution calculation at a frequency point is relatively cheap. Therefore, the spectral solver can be less effective for the analyses with a small number of frequency points (when the setup phase dominates), but it can be very effective for the same model when more frequency points are used.

The figure below shows the performance benefits from using the new spectral modal solver over the conventional (default) solver that solves the system of equations with a dense complex matrix and multiple right hand sides at every frequency point.



It compares the wall-time ratios for a steady-state dynamic analysis using the conventional and the new spectral solvers for the following models:

Model		DOF	Load Cases	Eigenmodes	Frequency Points	Damping Properties	Cores
1	Body-in-white	3.6 million	120	12200	1000	Material structural damping; no viscous damping	40

Model		DOF	Load Cases	Eigenmodes		Frequency Points	Damping Properties	Cores
2	Full vehicle	9.7 million	165	8800 uncoupled eigenmodes including 1000 acoustic modes		1000 2000	Material structural damping; viscous damping — 6 dashpots	40
3	Body-in-white	35.6 million	21	33800		1000	Material structural damping; viscous damping —70 dashpots	40 56
4	Body-in-trim	2.6 million	110	11000+	Uncoupled structural and acoustic eigenmodes Fully coupled structural-acoustic eigenmodes	1000	Material structural damping; viscous damping — 6 dashpots	
5	Fuselage model	48.3 million	120	21300		1000	Material structural damping; no viscous damping	40 56

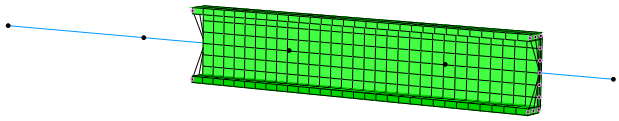
Beam-Based Submodeling from Global Beam Elements to Shell Elements

You can now perform beam-to- shell submodeling.

Benefits: Using shell elements in a submodel allows you to gain more detailed insight on results of interest for a region of a thin-walled beam that is modeled with beam elements in a global model.

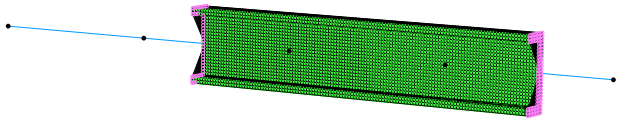
Beam-to-shell submodeling is often a good strategy to study details in one or more local regions if a global model uses beam elements.

The figure below shows a local shell-element model superimposed with beam elements from a global model. The shell thickness is rendered in the figure.



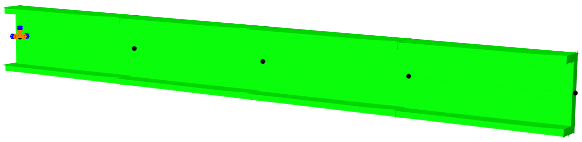
This simple example is provided to demonstrate the concept; typically, the global model would be larger and more complex than shown here.

Beam-to-shell submodeling is similar to the beam-to-solid submodeling capability introduced in Abaqus 2023 FD03; in both cases distributing couplings are used to impose conditions from the global model at submodel cuts. For comparison, the figure below shows a local solid-element model superimposed with beam elements from a global model.

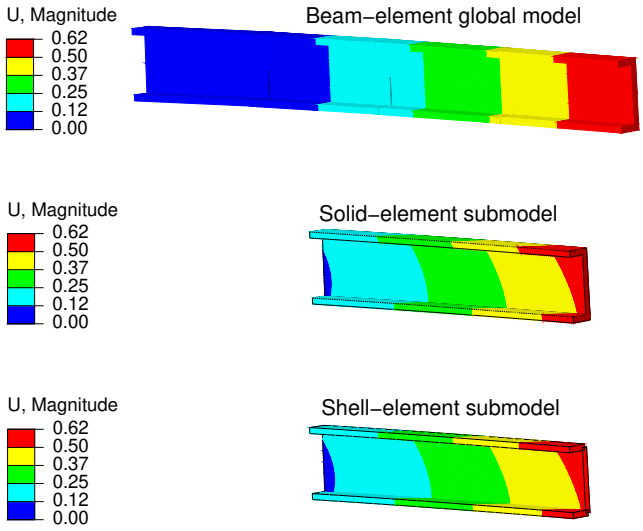


Representations of distributing couplings at submodel cuts are shown in both figures above. Reference nodes of these distributing couplings are used to impose conditions from the global model on the submodel.

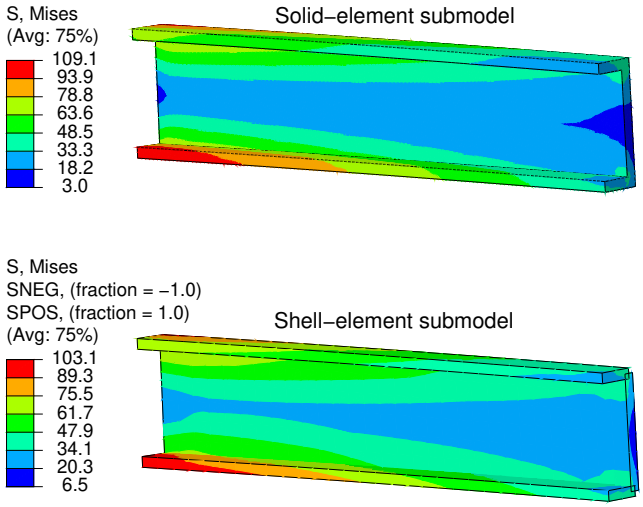
The figure below shows the global model with the beam cross-sections rendered.



The figure below provides a comparison of displacement contour plots for the beam-element global model and each of the submodels.



The figure below shows Mises stress contour plots for each of the submodels.



For more information, see *General Submodeling*.

2024 GA

Python 3

Abaqus Python is now based on Python 3.10 instead of Python 2.7. For more information about this major change and what it means for Abaqus users, visit *go.3ds.com/AbaqusPython*.

Benefits: This update brings Abaqus Python into conformance with a version of Python that is supported in the open-source community.

Formation of Tie Constraints in Import Analysis

In an Abaqus/Explicit import analysis, Abaqus/Explicit now forms tie constraints using the original instead of the updated configuration if the definitions of both the main and secondary surfaces to be tied are unchanged from the original analysis.

Benefits: The new protocol makes it easier to establish initial equilibrium due to tie constraints in the imported model.
The new protocol prevents the formation of new tie constraints for nodes on the surfaces that were previously untied. Such new tie constraints, if formed, can adversely affect the consistency of the imported model. For example, if the previously untied nodes are in a state of contact, tying these nodes results in deletion of the contact state associated with the nodes.

The new protocol also guarantees formation of the same tie constraints for the surfaces in the import analysis as in the original analysis. It is used regardless of whether you specify the original or the updated configuration as the reference configuration for import.

For more information, see *About Transferring Results between Abaqus Analyses*.

Adjoint Sensitivity Analysis for Sizing Optimization in Direct Steady-State Dynamic Analysis

You can now compute adjoint sensitivities for select node responses defined over node sets for a direct steady-state dynamic analysis to perform sizing optimization.

Benefits: You can now optimize for the steady-state response (direct-solution) of a structure subjected to harmonic excitations.

Abaqus/Standard now supports sizing optimization for select design variables (beam radii and shell thicknesses) in direct steady-state dynamic analyses for select design responses (nodal displacements/velocities/accelerations/rotations). You can define the range of frequencies over which you wish to compute the sensitivities for a particular design response.

You can use this new capability only for jobs run on a single domain and for node sets with nodes corresponding only to independent degrees of freedom.

For more information, see *Adjoint Design Sensitivity Analysis Overview*.

Analysis Procedures

In this section:

- *Introduction*
- *Static Stress/Displacement Analysis*
- *Dynamic Stress/Displacement Analysis*
- *Steady-State Transport Analysis*
- *Heat Transfer and Thermal-Stress Analysis*
- *Electromagnetic Analysis*
- *Electrochemical Analysis*
- *Coupled Pore Fluid Flow and Stress Analysis*
- *Mass Diffusion Analysis*
- *Coupled Acoustic-Structural Analysis*
- *Abaqus/Aqua Analysis*
- *Annealing*
- *One-Step Inverse Analysis*

Introduction

In this section:

- *Solving Analysis Problems*
- *Defining an Analysis*
- *General and Perturbation Procedures*
- *Multiple Linear Load Case Analysis*
- *Multiple Nonlinear Load Case Analysis*
- *Step Cycling*
- *Direct Linear Equation Solver*
- *Iterative Linear Equation Solver*

Solving Analysis Problems

Overview

A large class of stress analysis problems can be solved with Abaqus/Standard and Abaqus/Explicit. A fundamental division of such problems is into static or dynamic response; dynamic problems are those in which inertia effects are significant.

An analysis problem history is defined using steps in Abaqus (*Defining an Analysis*). For each step you choose an analysis procedure, which defines the type of analysis to be performed during the step. The available analysis procedures are listed below and described in more detail in the referenced sections.

Abaqus provides multiphysics capabilities using built-in fully coupled procedures, sequential coupling, and co-simulation as solution techniques for multiphysics simulation. An extensive selection of additional analysis techniques that provide powerful tools for performing your Abaqus analyses more efficiently and effectively is available; see *Analysis Techniques*.

References:

- *Analysis Techniques*
- *Coupled Pore Fluid Flow and Stress Analysis*
- *Dynamic Stress/Displacement Analysis*
- *Electromagnetic Analysis*
- *Heat Transfer and Thermal-Stress Analysis*
- *Static Stress/Displacement Analysis*

Abaqus/Standard Analysis

Abaqus/Standard offers complete flexibility in making the distinction between static and dynamic response; the same analysis can contain several static and dynamic phases. Thus, a static preload might be applied, and then the linear or nonlinear dynamic response computed (as in the case of vibrations of a component of a rotating machine or the response of a flexible offshore system that is initially moved to an equilibrium position subject to buoyancy and steady current loads and then is excited by wave loading). Similarly, the static solution can be sought after a dynamic event (by following a dynamic analysis step with a step of static loading). See *Static Stress/Displacement Analysis* and *Dynamic Stress/Displacement Analysis* for information on these types of procedures. In addition to static and dynamic stress analysis, Abaqus/Standard offers the following analysis types:

- *Steady-State Transport Analysis*
- *Heat Transfer and Thermal-Stress Analysis*
- *Electromagnetic Analysis*
- *Coupled Pore Fluid Flow and Stress Analysis*
- *Mass Diffusion Analysis*
- *Coupled Acoustic-Structural Analysis*
- *Abaqus/Aqua Analysis*

Abaqus/Explicit Analysis

Abaqus/Explicit solves dynamic response problems using an explicit direct-integration procedure. See *Dynamic Stress/Displacement Analysis* for more information on the explicit dynamic procedures available in Abaqus. Abaqus/Explicit also provides heat transfer, acoustic, and annealing analysis capabilities: see *Heat Transfer and Thermal-Stress Analysis*, *Coupled Acoustic-Structural Analysis*, and *Annealing* for details.

Multiphysics Analyses

Multiphysics is a coupled approach in the numerical solution of multiple interacting physical domains. Abaqus provides built-in fully coupled procedures, sequential coupling, and co-simulation as solution techniques for multiphysics simulation.

Built-in Fully Coupled Procedures

Native Abaqus multiphysics capabilities solve the physics by adding degrees of freedom representing each of the physical fields and using a single solver. Abaqus provides the following built-in fully coupled procedures to solve multidisciplinary simulations, where all physics fields are computed by Abaqus:

- *Fully Coupled Thermal-Stress Analysis*
- *Coupled Thermal-Electrical Analysis*
- *Fully Coupled Thermal-Electrical-Structural Analysis*
- *Piezoelectric Analysis* (electrical and mechanical coupling)
- *Eddy Current Analysis* (electromagnetic)
- *Coupled Pore Fluid Diffusion and Stress Analysis*
- *Coupled Acoustic-Structural Analysis*
- *Eulerian Analysis*

Sequential Coupling

A sequentially coupled multiphysics analysis can be used when the coupling between one or more of the physical fields in a model is only important in one direction. A common example is a thermal-stress analysis in which the temperature field does not depend strongly on the stress field. A typical sequentially coupled thermal-stress analysis consists of two Abaqus/Standard runs: a heat transfer analysis and a subsequent stress analysis.

You can perform sequentially coupled multiphysics analyses in Abaqus/Standard as described in the following sections:

- *Predefined Fields for Sequential Coupling*
- *Sequentially Coupled Thermal-Stress Analysis*
- *Predefined Loads for Sequential Coupling*

Co-Simulation

The co-simulation technique is a multiphysics capability for run-time coupling of Abaqus and another analysis program. An Abaqus analysis can be coupled to another Abaqus analysis or to a third-party analysis program to perform multidisciplinary simulations and multidomain (multimodel) coupling.

The co-simulation technique is described in the following sections:

- *About Co-Simulation*

- *Preparing an Abaqus Analysis for Co-Simulation*
- *Structural-to-Structural Co-Simulation*
- *Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation*
- *Executing a Co-Simulation*
- *System-Level Modeling between Logical and Physical Interactions*

Defining an Analysis

Overview

An analysis is defined in Abaqus by:

- dividing the problem history into steps;
- specifying an analysis procedure for each step; and
- prescribing loads, boundary conditions, and output requests for each step.

Abaqus distinguishes between general analysis steps and linear perturbation steps, and you can include multiple steps in your analysis. You can control how prescribed conditions are applied throughout each step. In addition, you can specify:

- the incrementation scheme used for controlling the solution,
- the matrix storage and solution scheme in Abaqus/Standard, and
- the precision level of the Abaqus/Explicit executable.

References:

• *Loads*

Defining an Analysis

An analysis in Abaqus is defined using steps, analysis procedures, and optional history data.

Defining Steps

A basic concept in Abaqus is the division of the problem history into *steps*. A step is any convenient phase of the history, such as thermal transient, creep hold, or dynamic transient step. In its simplest form a step can be a static analysis in Abaqus/Standard of a load change from one magnitude to another. You can provide a description of each step that appears in the data (`.dat`) file; this description is for convenience only.

The step definition includes the type of analysis to be performed and optional history data, such as loads, boundary conditions, and output requests.

Input File Usage: Use the first option to begin a step and the second option to end a step:

**STEP*
**END STEP*

The optional data lines on the **STEP* option can be used to specify the step description. The first data line given appears in the data (`.dat`) file.

Abaqus/CAE Usage: Step module: **Create Step**: **Description**

Specifying the Analysis Procedure

For each step you choose an *analysis procedure*. This choice defines the type of analysis to be performed during the step, such as static stress analysis, dynamic stress analysis, eigenvalue buckling, and transient heat transfer analysis. The available analysis procedures are described in *Solving Analysis Problems*. Only one procedure is allowed per step.

Input File Usage: The procedure definition option must immediately follow the **STEP* option.

Abaqus/CAE Usage: Step module: **Create Step**: choose the procedure type

Prescribing Loads, Boundary Conditions, and Output Requests

The step definition includes optional history data, such as loads, boundary conditions, and output requests, as defined in *History Data*. For more information, see *Boundary Conditions* , *Loads*, and *About Output*.

Details for prescribing these conditions are discussed in the individual procedure sections.

Input File Usage: The optional history data are defined following the procedure definition within a **STEP* block.

Abaqus/CAE Usage: You define history data (step-dependent objects) in the Interaction module, Load module, and Step module.

General Analysis Steps Versus Linear Perturbation Steps

There are two kinds of steps in Abaqus: general analysis steps, which can be used to analyze linear or nonlinear response, and linear perturbation steps, which can be used only to analyze linear problems. General analysis steps can be included in an Abaqus/Standard or Abaqus/Explicit analysis; linear perturbation analysis steps are available only in Abaqus/Standard. In Abaqus/Standard linear analysis is always considered to be linear perturbation analysis about the state at the time when the linear analysis procedure is introduced. This linear perturbation approach allows general application of linear analysis techniques in cases where the linear response depends on preloading or on the nonlinear response history of the model. See *General and Perturbation Procedures* for more details.

Multiple Load Case Analysis

In general analysis steps Abaqus/Standard calculates the solution for a single set of applied loads. This is also the default for linear perturbation steps. However, for static, direct steady-state dynamic, and SIM-based steady-state dynamic linear perturbation steps it is possible to find solutions for multiple load cases. See *Multiple Linear Load Case Analysis* for a description of this capability.

Multiple Steps

The analysis procedure can be changed from step to step in any meaningful way, so you have great flexibility in performing analyses. Since the state of the model (stresses, strains, temperatures, etc.) is updated throughout all general analysis steps, the effects of previous history are always included in the response in each new analysis step. Thus, for example, if natural frequency extraction is performed after a geometrically nonlinear static analysis step, the preload stiffness is included. Linear perturbation steps have no effect on subsequent general analysis steps.

The most obvious reason for using several steps in an analysis is to change analysis procedure type. However, several steps can also be used as a matter of convenience—for example, to change output requests, contact pairs in Abaqus/Explicit, boundary conditions, or loading (any information specified as history, or step-dependent, data). Sometimes an analysis may have progressed to a point where the present step definition needs to be modified. Abaqus provides for this contingency with the restart capability, where a step can be terminated prematurely and a new step can be defined for the problem continuation (see *Restarting an Analysis*).

Optional history data (see *Abaqus Model Definition*) prescribing the loading, boundary conditions, output controls, and auxiliary controls remains in effect for all subsequent general analysis steps, including those that are defined in a restart analysis, until they are modified or reset. Abaqus compares all loads and boundary conditions specified in a step with the loads and boundary conditions in effect during the previous step to ensure consistency and continuity. This comparison is expensive if the number of individually specified loads and boundary conditions is very large. Hence, the number of individually specified loads and boundary conditions should be minimized, which can usually be done by using element and node sets instead of individual elements and nodes. For linear perturbation steps only the output controls are continued from one linear perturbation step to the next if there are no intermediate general analysis steps and the output controls are not redefined (see *About Output*).

Within Abaqus/Standard or Abaqus/Explicit, any combination of available procedures can be used from step to step. However, Abaqus/Standard and Abaqus/Explicit procedures cannot be used in the same analysis. See *About Transferring Results between Abaqus Analyses* for information on importing results from one type of analysis to another.

Defining Time Varying Prescribed Conditions

By default, Abaqus assumes that external parameters, such as load magnitudes and boundary conditions, are constant (step function) or vary linearly (ramped) over a step, depending on the analysis procedure, as shown in *Table 1*. Some exceptions in Abaqus/Standard are discussed below.

Table 1: Default amplitude variations for time domain procedures.

Procedure	Default amplitude variation
Coupled pore fluid diffusion/stress (steady-state)	Ramp
Coupled pore fluid diffusion/stress (transient)	Step
Coupled thermal-electrical (steady-state)	Ramp
Coupled thermal-electrical (transient)	Step
Direct-integration dynamic	Step (exception: Ramp if quasi-static application type is specified)
Fully coupled thermal-electrical-structural in Abaqus/Standard (steady-state)	Ramp
Fully coupled thermal-electrical-structural in Abaqus/Standard (transient)	Step
Fully coupled thermal-stress in Abaqus/Standard (steady-state)	Ramp
Fully coupled thermal-stress in Abaqus/Standard (transient)	Step
Fully coupled thermal-stress in Abaqus/Explicit	Step
Magnetostatic	Ramp
Mass diffusion (steady-state)	Ramp
Mass diffusion (transient)	Step
Quasi-static	Step
Static	Ramp
Steady-state transport	Ramp

Procedure	Default amplitude variation
Transient eddy current	Step
Transient modal dynamic	Step
Uncoupled heat transfer	Ramp
Uncoupled heat transfer (transient)	Step

No default amplitude variation is defined for a direct cyclic analysis step; for each applied load or boundary condition, the amplitude must be defined explicitly.

Additional Default Amplitude Variations in Abaqus/Standard

For displacement or rotation degrees of freedom prescribed in Abaqus/Standard using displacement-type boundary conditions or displacement-type connector motions, the default amplitude variation is a ramp function for all procedure types. However, when using velocity-type boundary conditions or velocity-type connector motions in a static step or in a quasi-static dynamic step, the default amplitude is a step function.

For motions prescribed using a predefined displacement field, the default amplitude variation is a ramp function for all procedure types; the default amplitude is a step function when using a predefined velocity field for all procedures except steady-state transport.

The default amplitude variation is a step function for fluid flux loading in all procedure types.

When a displacement or rotation boundary condition is removed, the corresponding reaction force or moment is reduced to zero according to the amplitude defined for the step. When film or radiation loads are removed, the variation is always a step function.

Prescribing Nondefault Amplitude Variations

You can define complicated time variations of loadings, boundary conditions, and predefined fields by referring to an amplitude curve in the prescribed condition definition (see *Amplitude Curves*). User subroutines are also provided in Abaqus/Standard and Abaqus/Explicit for coding general loadings (see *About User Subroutines and Utilities*).

In Abaqus/Standard you can change the default amplitude variation for a step (except the removal of film or radiation loads, as noted above).

Input File Usage: In Abaqus/Standard use the following option to change the default amplitude variation for a step:

**STEP*, AMPLITUDE=STEP or RAMP

Abaqus/CAE Usage: In Abaqus/Standard use the following input to change the default amplitude variation for a step:

Step module: step editor: **Other**: **Default load variation with time: Instantaneous** or **Ramp linearly over step**

Boundary Conditions in Abaqus/Explicit

Boundary conditions applied during an explicit dynamic response step should use appropriate amplitude references to define the time variation. If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference is ignored, and a zero velocity boundary condition is enforced.

Prescribing Nondefault Amplitude Variations in Transient Procedures in Abaqus/Standard

The default amplitude is a step function for transient analysis procedures (fully coupled thermal-stress, fully coupled thermal-electrical-structural, coupled thermal-electrical, direct-integration dynamic, uncoupled heat transfer, and mass diffusion). You should exercise care when the nondefault ramp amplitude variation is specified for transient analysis procedures since unexpected results might occur. For example, if a step of a transient heat transfer analysis uses the ramp amplitude variation and temperature boundary conditions are removed in a subsequent step, the reaction fluxes generated in the previous step are instantaneously set to zero.

Incrementation

Each step in an Abaqus analysis is divided into multiple increments. In most cases you have two choices for controlling the solution: automatic time incrementation or user-specified fixed time incrementation. Automatic incrementation is recommended for most cases. The methods for selecting automatic or direct incrementation are discussed in the individual procedure sections.

The issues associated with time incrementation in Abaqus/Standard and Abaqus/Explicit are quite different, since time increments are generally much smaller in Abaqus/Explicit.

Incrementation in Abaqus/Standard

In nonlinear problems Abaqus/Standard increments and iterates as required to analyze a step, depending on the severity of the nonlinearity. In transient cases with a physical time scale, you can provide parameters to indicate a level of accuracy in the time integration, and Abaqus/Standard chooses the time increments to achieve this accuracy. Direct user control is provided because it can sometimes save computational cost in cases where you are familiar with the problem and know a suitable incrementation scheme. Direct control can also occasionally be useful when automatic control has trouble with convergence in nonlinear problems.

Specifying the Maximum Number of Increments

You can define the upper limit to the number of increments in an Abaqus/Standard analysis. In a direct cyclic analysis procedure, this upper limit should be set to the maximum number of increments in a single loading cycle. The default is 100. The analysis stops if this maximum is exceeded before the complete solution for the step has been obtained. To arrive at a solution, it is often necessary to increase the number of increments allowed by defining a new upper limit.

Input File Usage: **STEP*, INC=*n*

Abaqus/CAE Usage: Step module: step editor: **Incrementation**: **Maximum number of increments**

Extrapolation of the Solution

In nonlinear analyses Abaqus/Standard uses extrapolation to speed up the solution. Extrapolation refers to the method used to determine the first guess to the incremental solution. The guess is determined by the size of the current time increment and by whether linear, displacement-based parabolic, velocity-based parabolic, or no extrapolation of the previously attained history of each solution variable is chosen. Displacement-based parabolic extrapolation is not relevant for Riks analyses, and velocity-based parabolic extrapolation is available only for direct-integration dynamic procedures. Linear extrapolation (the default for all procedures other than a direct-integration dynamic procedure using the transient fidelity application setting) uses 100% extrapolation (1% for the Riks method) of the previous incremental solution at the start of each increment to begin the nonlinear equation solution for the next increment. No extrapolation is used in the first increment of a step.

In some cases extrapolation can cause Abaqus/Standard to iterate excessively; some common examples are abrupt changes in the load magnitudes or boundary conditions and if unloading occurs as a result of cracking (in concrete models) or buckling. In such cases you should suppress extrapolation.

Displacement-based parabolic extrapolation uses two previous incremental solutions to obtain the first guess to the current incremental solution. This type of extrapolation is useful in situations when the local variation of the solution with respect to the time scale of the problem is expected to be quadratic, such as the large rotation of structures. If parabolic extrapolation is used in a step, it begins after the second increment of the step: the first increment employs no extrapolation, and the second increment employs linear extrapolation. Consequently, slower convergence rates may occur during the first two increments of the succeeding steps in a multistep analysis.

Velocity-based parabolic extrapolation uses the previous displacement incremental solution to obtain the first guess to the current incremental solution. It is available only for direct-integration dynamic procedures, and it is the default if the transient fidelity application setting is specified as part of this procedure (see *Implicit Dynamic Analysis Using Direct Integration*). This type of extrapolation is useful in situations with smooth solutions—i.e., when velocities do not display so called "saw tooth" patterns—and in such cases it may provide a better first guess than other extrapolations. If velocity-based parabolic extrapolation is used in a step, it begins after the first increment of the step; the first increment employs initial velocities.

Input File Usage: Use the following option to choose linear extrapolation:

**STEP*, EXTRAPOLATION=LINEAR (default for all procedures other than a direct-integration dynamic procedure using the transient fidelity application setting)

Use the following option to choose displacement-based parabolic extrapolation:

**STEP*, EXTRAPOLATION=PARABOLIC

Use the following option to choose velocity-based parabolic extrapolation:

**STEP*, EXTRAPOLATION=VELOCITY PARABOLIC (default for a direct-integration dynamic procedure using the transient fidelity application setting)

Use the following option to choose no extrapolation:

**STEP*, EXTRAPOLATION= NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Extrapolation of previous state at start of each increment: Linear**, **Parabolic**, **Velocity parabolic**, **None**, or **Analysis product default**

Incrementation in Abaqus/Explicit

The time increment used in an Abaqus/Explicit analysis must be smaller than the stability limit of the central-difference operator (see *Explicit Dynamic Analysis*); failure to use a small enough time increment results in an unstable solution. Although the time increments chosen by Abaqus/Explicit generally satisfy the stability criterion, user control over the size of the time increment is provided to reduce the chance of a solution going unstable. The small increments characteristic of an explicit dynamic analysis product make Abaqus/Explicit well suited for nonlinear analysis.

Controlling the Analysis Based on the Simulation State in Abaqus/Standard

You can monitor one or more simulation states with sensors (see *Defining Sensors*) during the analysis and take an action, such as terminating the current step or the entire analysis, when the sensor value reaches a specified

threshold criterion. In addition, you can choose to modify the time increment as the simulation state approaches the threshold value or when the simulation state is within a predefined range of values.

The simulation state is quantified by monitoring the values of certain output variables at specific locations in the model. You must identify the specific output to be monitored as a "measure" of the simulation state (for example, an electric potential or a displacement at a given node) as a sensor (see *Defining Sensors*).

If you choose to affect the time increment as the threshold is approached to end the step or end the analysis, you can do so by:

- Specifying a fixed time increment value that Abaqus/Standard should use, once the condition for time increment refinement is met. The specified time increment will be used provided that it is lower than the time increment computed by the automatic time-incrementation algorithms in Abaqus/Standard, or
- Using the automatically determined time increment value, which is computed by Abaqus/Standard to ensure that the simulation state being monitored will meet the threshold value as closely as possible.

In addition, you can choose to refine the time increment during a certain interval of the analysis to monitor the state of the simulation closely within this interval. The interval of interest can be specified using start and end values of the sensor that is associated with the step control definition. When the value of the sensor is within the specified interval, the specified time increment is used, provided that it is lower than the time increment computed by the automatic time-incrementation algorithm in Abaqus/Standard. Once the value of the sensor is outside the specified limits, the time increment reverts back to the automatic time increments computed by Abaqus/Standard.

Step controls are subject to the following behaviors:

- There is no limit to the number of step controls you can specify in a given step,
- If multiple step controls with actions to end the current step or analysis are defined in a given step, the step control that reaches its associated threshold first is the one whose action is enforced, and
- If multiple step controls with time increment refinements are defined in a given step, Abaqus/Standard uses the smallest active specified time increment value, provided that it is lower than the time increment computed by the automatic time-incrementation

Step control definitions are subject to the following limitations:

- A given step control is active only in the step in which it is defined; there is no propagation of step controls to subsequent steps.
- Step controls can be defined in only general (non-perturbation) procedures in Abaqus/Standard.
- Automatic time incrementation control is not supported for coupled pore fluid diffusion and stress analysis (see *Coupled Pore Fluid Diffusion and Stress Analysis*), geostatic analysis (see *Geostatic Stress State*), quasi-static analysis (see *Quasi-Static Analysis*), and unstable collapse and postbuckling analysis (see *Unstable Collapse and Postbuckling Analysis*).

Input File Usage: Use the following option to use a sensor-based step control to stop a step or an analysis without refining the time increment:

**STEP CONTROL*, NAME=*step_control_name*, DT REFINEMENT=NO, ACTION=END STEP (or END ANALYSIS)
sensor_name, operate_criteria (max, min, etc), sensor_threshold_value

Use the following option to use a sensor-based step control to stop a step or an analysis and to refine the time increment while approaching the sensor threshold value:

**STEP CONTROL*, NAME=*step_control_name*, DT REFINEMENT=YES, ACTION=END STEP (or END ANALYSIS)

sensor_name, operate_criteria (max, min, etc), sensor_threshold_value, dt_refined_value, sensor_value_when_dt_refinement_starts,

Use the following option to use a sensor-based step control to stop a step or an analysis and to refine the time increment automatically to hit the sensor threshold value as closely as possible:

**STEP CONTROL*, NAME=*step_control_name*, DT REFINEMENT=AUTO, ACTION=END STEP (or END ANALYSIS)
sensor_name, operate_criteria (max, min, etc), sensor_threshold_value

Use the following option to use a sensor-based step control to refine the time increment between two sensor values:

**STEP CONTROL*, NAME=*step_control_name*, DT REFINEMENT=YES, ACTION=CONTINUE
sensor_name, , ,dt_refined_value, sensor_value_when_dt_refinement_starts, sensor_value_when_dt_refinement_ends

Severe Discontinuities in Abaqus/Standard

Abaqus/Standard distinguishes between regular, equilibrium iterations (in which the solution varies smoothly) and severe discontinuity iterations (SDIs) in which abrupt changes in stiffness occur. The most common of such severe discontinuities involve open-close changes in contact and stick-slip changes in friction. By default, Abaqus/Standard continues to iterate until the severe discontinuities are sufficiently small (or no severe discontinuities occur) and the equilibrium (flux) tolerances are satisfied. Alternatively, you can choose a different approach in which Abaqus/Standard continues to iterate until no severe discontinuities occur.

For contact openings with the default approach, a force discontinuity is generated when the contact force is set to zero, and this force discontinuity leads to force residuals that are checked against the time average force in the usual way, as described in *Convergence Criteria for Nonlinear Problems*. Similarly, in stick-to-slip transitions the frictional force is set to a lower value, which also leads to force residuals.

For contact closures a severe discontinuity is considered sufficiently small if the penetration error is smaller than the contact compatibility tolerance times the incremental displacement. The penetration error is defined as the difference between the actual penetration and the penetration following from the contact pressure and pressure-overclosure relation. In cases where the displacement increment is essentially zero, a “zero penetration” check is used, similar to the check used for zero displacement increments (see *Convergence Criteria for Nonlinear Problems*). The same checks are used for slip-to-stick transitions in Lagrange friction.

To make sure that sufficient accuracy is obtained for contact between hard bodies, it is also required that the estimated contact force error is smaller than the time average force times the contact force error tolerance. The estimated contact force error is obtained by multiplying the penetration by an effective stiffness. For hard contact this effective stiffness is equal to the stiffness of the underlying element, whereas for softened/penalty contact the effective stiffness is obtained by adding the compliance of the contact constraint and the underlying element.

Forcing the iteration process to continue until no severe discontinuities occur is the more traditional, conservative method. However, this method can sometimes lead to convergence problems, particularly in large problems with many contact points or situations where contact conditions are only weakly determined. In such cases excessive iteration may occur and convergence may not be obtained.

Input File Usage: **STEP*, CONVERT SDI=NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Convert severe discontinuity iterations: Off**

Matrix Storage and Solution Scheme in Abaqus/Standard

Abaqus/Standard generally uses Newton's method to solve nonlinear problems and the stiffness method to solve linear problems. In both cases the stiffness matrix is required. In some problems—for example, with Coulomb friction—this matrix is not symmetric. Abaqus/Standard automatically chooses whether a symmetric or unsymmetric matrix storage and solution scheme is used based on the model and step definition used. In some cases you can override this choice; the rules are explained below.

Usually it is not necessary to specify the matrix storage and solution scheme. The choice is available to improve computational efficiency in those cases where you judge that the default value is not the best choice. In certain cases where the exact tangent stiffness matrix is not symmetric, the extra iterations required by a symmetric approximation to the tangent matrix use less computer time than solving the nonsymmetric tangent matrix at each iteration. Therefore, for example, Abaqus/Standard invokes the symmetric matrix storage and solution scheme automatically in problems with Coulomb friction where every friction coefficient is less than or equal to 0.2, even though the resulting tangent matrix has some nonsymmetric terms. However, if any friction coefficient is greater than 0.2, Abaqus/Standard uses the unsymmetric matrix storage and solution scheme automatically since it may significantly improve the convergence history. This choice of the unsymmetric matrix storage and solution scheme considers changes to the friction model. Thus, if you modify the friction definition during the analysis to introduce a friction coefficient greater than 0.2, Abaqus/Standard activates the unsymmetric matrix storage and solution scheme automatically. In cases in which the unsymmetric matrix storage and solution scheme is selected automatically, you must explicitly turn it off if so desired; it is recommended to do so if friction prevents any sliding motions.

Input File Usage: **STEP*, UNSYMM=YES or NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Storage: Use solver default** or **Unsymmetric** or **Symmetric**

Rules for Using the Unsymmetric Matrix Storage and Solution Scheme

The following rules apply to matrix storage and solution schemes in Abaqus/Standard:

1. Since Abaqus/Standard provides eigenvalue extraction only for symmetric matrices, steps with eigenfrequency extraction or eigenvalue buckling prediction procedures always use the symmetric matrix storage and solution scheme. You cannot change this setting. In such steps Abaqus/Standard symmetrizes all contributions to the stiffness matrix.
2. In all steps except those with eigenfrequency extraction or eigenvalue buckling procedures, Abaqus/Standard uses the unsymmetric matrix storage and solution scheme when any of the following features are included in the model. You cannot change this setting.
   a. Heat transfer convection/diffusion elements (element types DCC*xxx*)
   b. General shell sections with unsymmetric section stiffness matrices (*Three-Dimensional Conventional Shell Element Library*)
   c. User-defined elements with unsymmetric element matrices (*User-Defined Elements*)
   d. User-defined material models with unsymmetric material stiffness matrices (*User-Defined Mechanical Material Behavior* or *User-Defined Thermal Material Behavior*)
   e. User-defined surface interaction models with unsymmetric interface stiffness matrices (*User-Defined Interfacial Constitutive Behavior*)
3. The following features all trigger the unsymmetric matrix storage and solution scheme for the step. You cannot change this setting.

a. Fully coupled thermal-stress analysis, except when a separated solution scheme is specified for the step (*Fully Coupled Thermal-Stress Analysis*)

b. Coupled thermal-electrical analysis, except when a separated solution scheme is specified for the step (*Coupled Thermal-Electrical Analysis*)

c. Fully coupled thermal-electrical-structural analysis (*Fully Coupled Thermal-Electrical-Structural Analysis*)

d. Coupled pore fluid diffusion/stress analysis with absorption or exsorption behavior (*Coupled Pore Fluid Diffusion and Stress Analysis*)

e. Coupled pore fluid diffusion/stress analysis (steady-state)

f. Coupled pore fluid diffusion/stress analysis (transient with gravity loading)

g. Mass diffusion analysis (*Mass Diffusion Analysis*)

h. Radiation view factor calculation controls (*Cavity Radiation in Abaqus/Standard*)

4. By default, the unsymmetric matrix storage and solution scheme is used for the complex eigenvalue extraction procedure. You can change this setting.
5. In all other cases you can control whether a symmetric or a full matrix storage and arithmetic solution is chosen. If you do not specify the matrix storage and solution scheme, Abaqus/Standard utilizes the value used in the previous general analysis step.
6. If you do not specify the matrix storage and solution scheme in the first step of an analysis, Abaqus/Standard chooses the unsymmetric scheme when any of the following are used:

   a. Any Abaqus/Aqua load type

   b. The concrete damaged plasticity material model

   c. Friction with a friction coefficient greater than 0.2

   The default value in the first step is the symmetric scheme for all other cases, except those covered by rules 2 and 3 above and for cases in which a friction coefficient is increased above 0.2 after the first step.
7. For radiative heat transfer surface interactions (*Thermal Contact Properties*), certain follower forces (such as concentrated follower forces or moments), three-dimensional finite-sliding analyses, any finite sliding in coupled pore fluid diffusion/stress analyses, and certain material models (particularly nonassociated flow plasticity models and concrete) introduce unsymmetric terms in the model's stiffness matrix. However, Abaqus/Standard does not automatically use the unsymmetric matrix storage and solution scheme when radiative heat transfer surface interactions are used. Specifying that the unsymmetric scheme should be used can sometimes improve convergence in such cases.
8. Coupled structural-acoustic and uncoupled acoustic analysis procedures in Abaqus/Standard generally use symmetric matrix storage and solution. Exceptions are the subspace-based steady-state dynamics or complex frequency procedures used for coupled structural-acoustic problems, where unsymmetric matrices are a consequence of the coupling procedure used in these cases. Using acoustic infinite elements or the acoustic flow velocity option triggers the unsymmetric matrix storage and solution scheme in Abaqus/Standard, except for natural frequency extraction using the Lanczos eigensolver, which uses symmetric matrix operations.
9. Unsymmetric matrix storage doubles the storage requirement for element operators and for the equation solver compared to symmetric storage. These are generally the largest components of the overall memory requirements for the analysis.

Precision Level of the Abaqus/Explicit Executable

You can choose a double-precision executable (with 64-bit word lengths) for Abaqus/Explicit on machines with a default, single-precision word length of 32 bits (see *Abaqus/Standard and Abaqus/Explicit Execution*). Most new computers have 32-bit default word lengths even though they may have 64-bit memory addressing. The

single-precision executable typically results in a CPU savings of 20% to 30% compared to the double-precision executable, and single precision provides accurate results in most cases. Exceptions in which single precision tends to be inadequate include analyses that require greater than approximately 300,000 increments, have typical nodal displacement increments less than 10^{-6} times the corresponding nodal coordinate values, include hyperelastic materials, or involve multiple revolutions of deformable parts; the double-precision executable is recommended in these cases (for example, see *Simulation of propeller rotation*).

You can also run only a part of Abaqus/Explicit using double precision, while using single precision for the rest (see *Abaqus/Standard and Abaqus/Explicit Execution*). These options are described below.

- If **double**=`explicit` is used or the **double** option is specified without a value, the Abaqus/Explicit analysis runs in double precision, while the packager runs in single precision. While this choice would satisfy higher precision needs in most analyses, the data are written to the state (`.abq`) file in single precision. Moreover, analysis-related computations performed in the packager are still run in single precision. Thus, new steps, restart, and import analyses begin from data that are stored/computed in single precision despite the fact that calculations during the step are performed in double precision. Thus, in general,you can expect somewhat noisy solutions at the beginning of the first step, at step transitions, on restart, and after import.
- If **double**=`both` is used, both the Abaqus/Explicit packager and analysis run in double precision. This is the most expensive option but ensures the highest overall execution precision. Analysis database floating point data are written to the state (`.abq`) file at the end of packager or of a given step in double precision, thus ensuring in most cases the smoothest transition at step boundaries, on restart, and after an import.
- There may be cases where the default single precision analysis is inadequate, while the **double**=`both` option is too expensive. These are typically models that have complex links of constraints (such as a complex mechanism with connector elements, complex combinations of distributed/kinematic couplings, tie constraints and multi-point constraints, or interactions of such constraints with boundary conditions). For such models it is desirable to solve only the constraints in the model in double precision while the rest of the model is solved in single precision. This combination gives the desired accuracy of the solution while increasing performance compared to a full double precision analysis.
- If **double**=`constraint` is used, the constraint packager and constraint solver are executed in double precision, while the remainder of the Abaqus/Explicit packager and analysis are executed in single precision.
- If **double**=`off` is used or the **double** option is omitted (default), both the Abaqus/Explicit packager and the analysis run in single precision. The **double**=`off` option is useful when you want to override the setting in the environment file.

The significance of the precision level is indicated by comparing the solutions obtained with single and double precision. If no significant difference is found between single- and double-precision solutions for a particular model, the single-precision executable can be deemed adequate.

General and Perturbation Procedures

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Defining an Analysis*
- *Linear and nonlinear procedures*

Overview

An analysis step during which the response can be either linear or nonlinear is called a *general* analysis step. An analysis step during which you activate a perturbation procedure that determines the response about a base state due to perturbation loads and boundary conditions is called a *linear perturbation* analysis step. General analysis steps can be included in an Abaqus/Standard or Abaqus/Explicit analysis; linear perturbation analysis steps are available only in Abaqus/Standard.

A clear distinction is made in Abaqus/Standard between general analysis and linear perturbation analysis steps. Loading conditions are defined differently for the two cases, time measures are different, and the results should be interpreted differently. These distinctions are defined in this section.

Abaqus/Standard treats a perturbation analysis as a perturbation about a preloaded, predeformed state. Abaqus/Foundation, a subset of Abaqus/Standard, is limited entirely to perturbation analysis but does not allow preloading or predeformed states. You cannot run an Abaqus/Foundation analysis in a restart job.

General Analysis Steps

A general analysis step is one in which the effects of any nonlinearities present in the model can be included. The starting condition for each general step is the ending condition from the last general step, with the state of the model evolving throughout the history of general analysis steps as it responds to the history of loading. If the first step of the analysis is a general step, the initial conditions for the step can be specified directly (*Initial Conditions*).

Abaqus always considers *total time* to increase throughout a general analysis. Each step also has its own *step time*, which begins at zero in each step. If the analysis procedure for the step has a physical time scale, as in a dynamic analysis, step time must correspond to that physical time. Otherwise, step time is any convenient time scale—for example, 0.0 to 1.0—for the step. The step times of all general analysis steps accumulate into total time. Therefore, if an option such as creep (available only in Abaqus/Standard) whose formulation depends on total time is used in a multistep analysis, any steps that do not have a physical time scale should have a negligibly small step time compared to the steps in which a physical time scale does exist.

Sources of Nonlinearity

Nonlinear stress analysis problems can contain up to three sources of nonlinearity: material nonlinearity, geometric nonlinearity, and boundary nonlinearity.

Material Nonlinearity

Abaqus offers models for a wide range of nonlinear material behaviors (see *Combining Material Behaviors*). Many of the materials are history dependent: the material's response at any time depends on what has happened to it at previous times. Thus, the solution must be obtained by following the actual loading sequence. The general analysis procedures are designed with this in view.

Geometric Nonlinearity

It is possible in Abaqus to define a problem as a "small-displacement" analysis, which means that geometric nonlinearity is ignored in the element calculations—the kinematic relationships are linearized. By default, large displacements and rotations are accounted for in contact constraints even if the small-displacement element formulations are used for the analysis; i.e., a large-sliding contact tracking algorithm is used (see *Contact Formulations in Abaqus/Standard* and *Contact Formulations for Contact Pairs in Abaqus/Explicit*). The elements in a small-displacement analysis are formulated in the reference (original) configuration, using original nodal coordinates. The errors in such an approximation are of the order of the strains and rotations compared to unity. The approximation also eliminates any possibility of capturing bifurcation buckling, which is sometimes a critical aspect of a structure's response (see *Unstable Collapse and Postbuckling Analysis*). You must consider these issues when interpreting the results of such an analysis.

The alternative to a "small-displacement" analysis in Abaqus is to include large-displacement effects. In this case most elements are formulated in the current configuration using current nodal positions. Elements therefore distort from their original shapes as the deformation increases. With sufficiently large deformations, the elements may become so distorted that they are no longer suitable for use; for example, the volume of the element at an integration point may become negative. In this situation Abaqus will issue a warning message indicating the problem. In addition, Abaqus/Standard will cut back the time increment before making further attempts to continue the solution. Abaqus/Explicit also offers element failure models to allow elements that reach high strains to be removed from a model; see *Dynamic Failure Models* for details.

For each step of an analysis you specify whether a small- or large-displacement formulation should be used (i.e., whether geometric nonlinearity should be ignored or included). By default, Abaqus/Standard uses a small-displacement formulation and Abaqus/Explicit uses a large-displacement formulation. The default value for the formulation in an import analysis is the same as the value at the time of import. If a large-displacement formulation is used during any step of an analysis, it will be used in all following steps in the analysis; there is no way to turn it off.

Almost all of the elements in Abaqus use a fully nonlinear formulation. The exceptions are the cubic beam elements in Abaqus/Standard and the small-strain shell elements (those shell elements other than S3/S3R, S4, S4R, and the axisymmetric shells) in which the cross-sectional thickness change is ignored so that these elements are appropriate only for large rotations and small strains. Except for these elements, the strains and rotations can be arbitrarily large.

The calculated stress is the "true" (Cauchy) stress. For beam, pipe, and shell elements the stress components are given in local directions that rotate with the material. For all other elements the stress components are given in the global directions unless a local orientation (*Orientations*) is used at a point. For small-displacement analysis the infinitesimal strain measure is used, which is output with the strain output variable E; strain output specified with output variables LE and NE is the same as with E.

Input File Usage: Use the following option to specify that a large-displacement formulation should be used for the step:

**STEP*, NLGEOM=YES (default in Abaqus/Explicit)

Use the following option to specify that a small-displacement formulation should be used for the step:

**STEP*, NLGEOM=NO (default in Abaqus/Standard)

Omitting the NLGEOM parameter is equivalent to using the default value.

Abaqus/CAE Usage: Step module: **Create Step**: select any step type: **Basic**: **Nlgeom**: **Off** (for a small-displacement formulation) or **On** (for a large-displacement formulation)

Boundary Nonlinearity

Contact problems are a common source of nonlinearity in stress analysis—see *About Contact Interactions*. Other sources of boundary nonlinearity are nonlinear elastic springs, films, radiation, multi-point constraints, etc.

Loading

In a general analysis step the loads must be defined as total values. The rules for applying loads in a general, multistep analysis are defined in *About Loads*.

Incrementation

The general analysis procedures in Abaqus offer two approaches for controlling incrementation. Automatic control is one choice: you define the step and, in some procedures, specify certain tolerances or error measures. Abaqus then automatically selects the increment size as it develops the response in the step. Direct user control of increment size is the alternative approach, whereby you specify the incrementation scheme. The direct approach is sometimes useful in repetitive analyses with Abaqus/Standard, where you have a good "feel" for the convergence behavior of the problem. The methods for selecting automatic or direct incrementation are discussed in the individual procedure sections.

In nonlinear problems in Abaqus/Standard the challenge is always to obtain a convergent solution in the least possible computational time. In these cases automatic control of the time increment is usually more efficient because Abaqus/Standard can react to nonlinear response that you cannot predict ahead of time. Automatic control is particularly valuable in cases where the response or load varies widely through the step, as is often the case in diffusion-type problems such as creep, heat transfer, and consolidation. Ultimately, automatic control allows nonlinear problems to be run with confidence in Abaqus/Standard without extensive experience with the problem.

Strong nonlinearities typically do not present difficulties in Abaqus/Explicit because of the small time increments that are characteristic of an explicit dynamic analysis product.

Stabilization of Unstable Problems in Abaqus/Standard

Some static problems can be naturally unstable, for a variety of reasons.

Unconstrained Rigid Body Motions

Instability may occur because unconstrained rigid body motions exist. Abaqus/Standard may be able to handle this type of problem with automatic viscous damping (see *Generally Applicable Contact Controls in Abaqus/Standard*) when rigid body motions exist during the approach of two bodies that will eventually come into contact.

Input File Usage: Use one of the following options:

**CONTACT STABILIZATION*
**CONTACT CONTROLS*, STABILIZE

Abaqus/CAE Usage: Automatic viscous damping is not supported in Abaqus/CAE.

Localized Buckling Behavior or Material Instability

Instability may also be caused by localized buckling behavior or by material instability; such instabilities are especially significant when no time-dependent behavior exists in the material modeling. The static, general analysis procedures in Abaqus/Standard can stabilize this type of problem if you request it (see *Static Stress*

Analysis, *Quasi-Static Analysis*, *Steady-State Transport Analysis*, *Fully Coupled Thermal-Stress Analysis*, *Fully Coupled Thermal-Electrical-Structural Analysis*, or *Coupled Pore Fluid Diffusion and Stress Analysis*).

Input File Usage: Use one of the following options:

**STATIC*, STABILIZE
**VISCO*, STABILIZE
**STEADY STATE TRANSPORT*, STABILIZE
**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE
**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, STABILIZE
**SOILS*, CONSOLIDATION, STABILIZE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: *any valid step type*: **Basic**: **Use stabilization with dissipated energy fraction**

Linear Perturbation Analysis Steps

Linear perturbation analysis steps are available only in Abaqus/Standard (Abaqus/Foundation is essentially the linear perturbation functionality in Abaqus/Standard). The response in a linear analysis step is the linear perturbation response about the *base state*. The base state is the current state of the model at the end of the last general analysis step prior to the linear perturbation step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions (*Initial Conditions*). In Abaqus/Foundation the base state is always determined from the initial state of the model.

Linear perturbation analyses can be performed from time to time during a fully nonlinear analysis by including the linear perturbation steps between the general response steps. The linear perturbation response has no effect as the general analysis is continued. The step time of linear perturbation steps, which is taken arbitrarily to be a very small number, is never accumulated into the total time. A simple example of this method is the determination of the natural frequencies of a violin string under increasing tension (see *Vibration of a cable under tension*). The tension of the string is increased in several geometrically nonlinear analysis steps. After each of these steps, the frequencies can be extracted in a linear perturbation analysis step.

If geometric nonlinearity is included in the general analysis upon which a linear perturbation study is based, stress stiffening or softening effects and load stiffness effects (from pressure and other follower forces) are included in the linear perturbation analysis.

Load stiffness contributions are also generated for centrifugal and Coriolis loading. In direct steady-state dynamic analysis Coriolis loading generates an imaginary antisymmetric matrix. This contribution is accounted for currently in solid and truss elements only and is activated by using the unsymmetric matrix storage and solution scheme in the step.

Linear Perturbation Procedures

The following purely linear perturbation procedures are available in Abaqus/Standard:

- *Eigenvalue Buckling Prediction*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Complex Eigenvalue Extraction*
- *Transient Modal Dynamic Analysis*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*

- *Response Spectrum Analysis*
- *Random Response Analysis*
- *Time-Harmonic Analysis*

In addition, the following analysis techniques are treated as linear perturbation steps in an analysis:

- *Generating Substructures*
- *Generating Matrices as a Linear Analysis Step*

Except for these procedures and the static procedure (explained below), all other procedures can be used only in general analysis steps (in other words, they are not available with Abaqus/Foundation). All linear perturbation procedures except for the complex eigenvalue extraction procedure are available with Abaqus/Foundation.

Static Perturbation Analysis

A static perturbation stress analysis (*Static Stress Analysis*) can be conducted in Abaqus/Standard.

Input File Usage: Use both of the following options to conduct a static perturbation analysis:

**STEP*, PERTURBATION
**STATIC*

Omitting the PERTURBATION parameter on the **STEP* option implies that a general static analysis is required.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Static, Linear perturbation**

Contact within Static Perturbation Analysis

Two approaches are available for handling contact in a static perturbation analysis.

In the static linear perturbation procedure, which is the default, the open/closed status of each contact constraint is assumed to remain as it is in the base state. Further, points in contact (i.e., with a “closed” status) are also assumed to be sticking if friction is present except when a velocity differential is imposed by the motion of the reference frame or the transport velocity. In the latter case, slipping conditions are assumed regardless of the friction coefficient. Thus, by freezing contact status, the contact contributions and thereby the overall governing equations are imposed to be linear in the solution variables and result in a purely linear static perturbation analysis.

In the special case where all contact constraints in an analysis are modeled with the small sliding formulation and friction is absent, the non-default LCP solution technique (see *Linear Complementarity Problem (LCP) Solution Technique for Solving Contact Problems*) can be activated. The static LCP perturbation procedure treats contact in a nonlinear manner by allowing for contact status changes due to applied perturbation loads and boundary conditions. As a result, the actual set of points in contact (that can be different from base state) and their normal contact pressure values are computed as a part of the solution process during the perturbation analysis. The static LCP perturbation procedure, with the exception of nonlinearities from contact, shares its behavior with the static linear perturbation procedure in most respects. For example, the solution from the perturbation analysis including the (possibly) modified contact status is not carried over to subsequent steps.

Loading and Output

Load magnitudes (including the magnitudes of prescribed boundary conditions and predefined temperatures and fields) during a linear perturbation analysis step are defined as the magnitudes of the load perturbations only. Likewise, the value of any solution variable is output as the perturbation value only—the value of the variable in the base state is not included.

Multiple Load Case Analysis

Multiple load cases can be analyzed simultaneously for static, direct-solution steady-state dynamic and SIM-based steady-state dynamic (including subspace projection) linear perturbation steps. See *Multiple Linear Load Case Analysis* for a description of this capability.

Restrictions

A linear perturbation analysis is subject to the following restrictions:

- In a linear perturbation analysis, amplitude references (*Amplitude Curves*) can be used to specify loads and boundary conditions as functions of frequency (in a direct steady-state dynamic analysis) or to define loads and base motion as functions of frequency (in frequency-domain mode-based dynamic procedures) or as functions of time (in a transient modal dynamic analysis). If loads or boundary conditions are specified as functions of time in a linear perturbation analysis that has no time period (for example, in a static perturbation analysis), the amplitude value corresponding to time=0 will be used.
- A general implicit dynamic analysis (*Implicit Dynamic Analysis Using Direct Integration*) cannot be interrupted to perform perturbation analyses: before performing the perturbation analysis, Abaqus/Standard requires that the structure be brought into static equilibrium.
- During a linear perturbation analysis step, the model's response is defined by its linear elastic stiffness at the base state. For viscoelastic materials the elastic moduli at the base state is evaluated as described in *Material Options* for static procedures and in *Evaluating Frequency-Dependent Material Properties* for frequency-based procedures. Plasticity and other inelastic effects are ignored during a linear perturbation analysis step. For hyperelasticity (*Hyperelastic Behavior of Rubberlike Materials*) or hypoelasticity (*Hypoelastic Behavior*), the tangent elastic moduli in the base state are used. If cracking has occurred—for example, in the concrete model (*Concrete Smeared Cracking*)—the damaged elastic (secant) moduli are used.
- The default behavior where the contact state is not allowed to change applies to any linear perturbation step and not just to static linear perturbation steps (see *Contact within Static Perturbation Analysis*). This restriction does not apply to the LCP solution technique, which can be activated only within a static perturbation step.
- The effects of temperature and field variable perturbations are ignored for materials that are dependent on temperature and field variables. However, temperature perturbations will produce perturbations of thermal and field expansion strains (see *Computing Thermal Strains in Linear Perturbation Steps* and *Computing Field Expansion Strains in Linear Perturbation Steps*).

Multiple Linear Load Case Analysis

Products: Abaqus/Standard Abaqus/CAE

References:

- **LOAD CASE*
- **END LOAD CASE*
- *Load cases*

Overview

A multiple linear load case analysis:

- is used to study the linear responses of a structure subjected to distinct sets of loads, predefined temperature fields, and boundary conditions defined within a step (each set is referred to as a load case);
- can be much more efficient than an equivalent multiple perturbation step analysis;
- allows you to change mechanical loads, temperatures, and boundary conditions from linear load case to linear load case;
- includes the effects of the base state; and
- can be performed with static perturbation, direct-solution steady-state dynamic, and SIM-based steady-state dynamic analyses.

Linear Load Cases

A linear load case refers to a set of loads, temperatures, boundary conditions, and base motions comprising a particular loading condition. For example, in a simplified model the operational environment of an airplane might be broken into five linear load cases: (1) take-off, (2) climb, (3) cruise, (4) descent, and (5) landing. Often a linear load case is defined in terms of unit loads or prescribed boundary conditions, and a multiple linear load case analysis refers to the simultaneous solution for the responses of each linear load case in a set of such linear load cases. These responses can then be scaled and linearly combined during postprocessing to represent the actual loading environment. Other postprocessing manipulations on linear load cases are also common, such as finding the maximum von Mises stress among all linear load cases. These types of linear load case manipulations can be requested in the Visualization module of Abaqus/CAE (see the *Introduction*).

Using Multiple Linear Load Cases

A multiple linear load case analysis is conceptually equivalent to a multiple step analysis in which the linear load case definitions are mapped to consecutive perturbation steps. However, a multiple linear load case analysis is generally much more efficient than the equivalent multiple step analysis. The exception occurs when a large number of boundary conditions exist that are not common to all linear load cases (that is, degrees of freedom are constrained in one linear load case but not others). It is difficult to define what “large” is since it is model dependent. The relative performance of the two analysis methods can be assessed by performing a data check analysis for both the multiple linear load case analysis and the equivalent multiple step analysis. The data check analysis writes resource information for each step to the data file, including the maximum wavefront, number of floating point operations, and minimum memory required. If these numbers are noticeably larger for the multiple linear load case step compared to those across all steps of the equivalent multiple step analysis (the

number of floating point operations can be summed over all steps before comparing), the multiple step analysis is more efficient.

Although generally more efficient, the multiple linear load case analysis might consume more memory and disk space than an equivalent multiple step analysis. Thus, for large problems or problems with many load cases it is again advisable, as described above, to compare resource usage between the multiple linear load case analysis and the equivalent multiple step analysis. If resource requirements for the multiple linear load case analysis are deemed too large, consider dividing the linear load cases among a few steps. The resulting analysis (a hybrid of multiple linear load cases and multiple steps) requires fewer resources while retaining an efficiency advantage over an equivalent pure multiple step analysis.

Defining Linear Load Cases

You define a linear load case within a static perturbation, direct-solution steady-state dynamic, and SIM-based steady-state dynamic analyses. Linear load case definitions do not propagate to subsequent steps. Only the following types of prescribed conditions can be specified within a linear load case definition:

- Boundary conditions
- Concentrated loads
- Distributed loads
- Distributed surface loads
- Predefined temperature fields
- Inertia-based loads
- Base motions
- Output requests to the output database

Additional rules governing these prescribed conditions are described in the sections that follow. No other types of prescribed conditions can appear in a step that contains linear load case definitions. All other valid analysis components, such as output requests to the data file, must be specified outside the linear load case definitions.

Each linear load case definition is assigned a name for postprocessing purposes.

Input File Usage: Use the first option to begin a linear load case and the second option to end a linear load case:

**LOAD CASE*, NAME=*name*
**END LOAD CASE*

Use the following option for linear load cases that include distributed nodal pressure loading:

**LOAD CASE*, NAME=*name*, GENERATE
**END LOAD CASE*

Prescribed conditions specified within a linear load case definition apply only to that linear load case. In static perturbation and direct-solution steady-state dynamic analyses, prescribed conditions can be specified outside the linear load case definitions (in this case they apply to all linear load cases in the step).

Abaqus/CAE Usage: Load module: **Create Load Case**: **Name**: *name*

In Abaqus/CAE if a step contains linear load cases, all prescribed conditions in the step must be included in one or more linear load cases.

Procedures

Linear load cases can be defined only in perturbation steps with the following procedures:

- Static
- Direct-solution, steady-state dynamic
- SIM-based, steady-state dynamic

As with other perturbation steps, a multiple linear load case analysis includes the nonlinear effects of the previous general step (base state). The following analysis techniques are not supported in the context of a linear load case step:

- Restart from a particular linear load case
- Submodeling using results from other than the first linear load case in the global analysis
- Importing and transferring results
- Cyclic symmetry analysis
- Contour integrals
- Design sensitivity analysis

Boundary Conditions

Boundary conditions can be specified both outside and inside linear load case definitions in the same step. Specifying a boundary condition outside the linear load case definitions in a step is equivalent to including it in all linear load case definitions in the step (that is, the boundary condition is applied to all linear load cases). Unless any boundary conditions are removed in the perturbation step, the boundary conditions that are active in the base state will propagate to all linear load cases in the perturbation step. If any boundary condition is removed in a step with linear load cases (either outside or inside the linear load case definitions), the base state boundary conditions will not be propagated to any linear load case in the step. See *Boundary Conditions* for more information.

You should redefine identical boundary conditions between linear load cases as described in *Boundary Conditions*. You must apply constraints consistently using either the “type” (name) format or the degree of freedom “direct” format without changing the format between linear load cases. Otherwise, Abaqus treats the redefined boundary conditions as changing between linear load cases, which increases the computational cost of the analysis.



Note:

In Abaqus/CAE if a step contains linear load cases, all boundary conditions in the step must be included in one or more linear load cases. Boundary conditions can only be used with linear load cases in static perturbation and direct-solution steady-state dynamic analyses.

Loads

In static perturbation and direct-solution steady-state dynamic analyses, you can specify concentrated, distributed, and distributed surface loads both outside and inside linear load case definitions in the same step. The only exceptions are Coriolis and rotor dynamic loads, which cannot be specified inside linear load case definitions in a direct-solution steady-state dynamic step. These loads contribute to the left hand side of the system of element equations. Any changes to these loads within the linear load case definitions results in changes to the left hand side of the overall system of equations from linear load case to linear load case.

You can specify inertia relief loads either outside linear load case definitions or inside linear load case definitions in the same step but not both simultaneously. Specifying one of these load types outside the linear load case definitions in a step is equivalent to including it in all linear load case definitions in the step (that is, the loading is applied to all linear load cases).

Connector loads and connector motion are not supported for a linear load case analysis.

In SIM-based steady-state dynamic analyses concentrated, distributed, distributed surface loads, and base motion can be specified only inside linear load case definitions in the same step. Inertia relief loads are not supported.

Linear load cases cannot be used in models that include aqua loads (see *Abaqus/Aqua Analysis*).

As with any perturbation step, perturbation loads must be defined completely within the perturbation step (see *About Loads*).



Note:

In Abaqus/CAE if a step contains linear load cases, all loads in the step must be included in one or more linear load cases.

Predefined Fields

In static perturbation analyses you can specify predefined temperature fields both outside and inside linear load case definitions within the same step. Specifying temperature outside the linear load case definitions in a step is equivalent to including it in all linear load case definitions in the step. If a temperature field is specified at the same node both outside and inside a linear load case, the temperature definition inside the linear load case takes precedence, and the temperature definition outside the linear load case is discarded at this node.

You cannot specify field variables in a step with linear load cases.

Elements

Linear load cases cannot be used in models that include piezoelectric elements (see *Piezoelectric Analysis*).

Output

In a step containing one or more linear load cases, only selected field and history output requests to the output database and output requests to the data file are supported. Output requests to the results file are not supported. Output requests specified outside linear load case definitions apply to all linear load cases in a step. Output requests to the output database specified inside a specific linear load case definition apply only to that linear load case. Output requests to the data file are not supported inside a linear load case. Output requests inside a linear load case do not propagate to subsequent steps. For all other output requests, the step propagation rules are the same as for other perturbation steps (see *About Output*).

Element and energy history output variables are not available during a multiple linear load case analysis (see *Abaqus/Standard Output Variable Identifiers*). Additional restrictions apply for a SIM-based steady-state dynamic analysis; see *Using the SIM Architecture for Modal Superposition Dynamic Analyses* for more information.

The available field output corresponding to each linear load case is stored in a separate frame on the output database with the linear load case name included as a frame attribute. To distinguish between linear load cases for history output variables, the name of the linear load case is appended to the history variable name. The Visualization module of Abaqus/CAE and the Abaqus Scripting Interface (see *Using the Abaqus Scripting Interface to access an output database*) can be used to access and manipulate linear load case output.Abaqus/Standard does not perform consistency checks on the physical validity of the linear load case manipulations. For example, the linear superposition of two linear load cases, each with different boundary conditions, is allowed even though the combined results might not be physically meaningful.

Limitations

For frame elements, the temperature specified inside a linear load case definition is ignored.

Input File Template

```
*HEADING
…
*STEP, PERTURBATION
*STATIC or *STEADY STATE DYNAMICS, DIRECT
…
*OUTPUT, FIELD
…
*BOUNDARY
Data lines to specify boundary conditions for all linear load cases.
*DLOAD
Data lines to specify distributed loads for all linear load cases.
*CLOAD
Data lines to specify point loads for all linear load cases.
*DSLOAD
Data lines to specify distributed surface loads for all linear load cases.
*INERTIA RELIEF
Data lines to specify inertia relief loading directions.
(This option cannot be used inside linear load cases if it is used here.)
…
*LOAD CASE, NAME=name1
*BOUNDARY
Data lines to specify boundary conditions for first linear load case.
*DLOAD
Data lines to specify distributed loads for first linear load case.
*CLOAD
Data lines to specify point loads for first linear load case.
*DSLOAD
Data lines to specify distributed surface loads for first linear load case.
*INERTIA RELIEF
Data lines to specify inertia relief loading directions.
(This option cannot be used outside linear load cases if it is used here.)
*END LOAD CASE
*LOAD CASE, NAME=name2
Load and boundary condition options for second linear load case
*END LOAD CASE
…
Subsequent linear load case definitions
…
*END STEP
*STEP, PERTURBATION
*FREQUENCY, SIM or *FREQUENCY, EIGENSOLVER=AMS
*END STEP
…
*STEP, PERTURBATION
*STEADY STATE DYNAMICS
*LOAD CASE, NAME=name3
*BASE MOTION
Data lines to specify base motion for first linear load case.
*DLOAD
Data lines to specify distributed loads for first linear load case.
```

```
*CLOAD
Data lines to specify point loads for first linear load case.
*DSLOAD
Data lines to specify distributed surface loads for first linear load case.
*END LOAD CASE
*LOAD CASE, NAME=name4
Load and base motion options for second linear load case.
*END LOAD CASE
…
Subsequent linear load case definitions
…
*OUTPUT, HISTORY
…
*END STEP
*STEP, PERTURBATION
*STATIC
…
*OUTPUT, FIELD
…
*BOUNDARY
Data lines to specify boundary conditions for all linear load cases.
*LOAD CASE, NAME=name5, GENERATE
*DSLOAD
Data lines to specify distributed nodal pressure linear load cases on surfaces.
*END LOAD CASE
…
Subsequent linear load case definitions
…
*END STEP
*STEP, PERTURBATION
*STATIC
*TEMPERATURE
Data lines to specify temperature for all linear load cases.
*LOAD CASE, NAME=name6
*TEMPERATURE
Data lines to specify temperature for first linear load case.
*END LOAD CASE
…
Subsequent linear load case definitions
…
*END STEP
```

Multiple Nonlinear Load Case Analysis

Products: Abaqus/Standard

References:

- **MANIFEST*

Overview

A multiple nonlinear load case analysis:

- is used to study the nonlinear responses of a structure subjected to distinct load histories consisting of one or more steps (each load history is referred to as a nonlinear load case);
- can include the effects of a base state common to all nonlinear load cases;
- allows results to be combined in a single set of output files; and
- can be much more efficient than running each nonlinear load case as a separate analysis.

Nonlinear Load Cases

A nonlinear load case refers to a set of steps comprising a particular load history. For example, the operational environment of an airplane might be broken into three nonlinear load cases: (1) slow ascent and then bank left, (2) rapid ascent and then bank right, and (3) descent, where load cases (1) and (2) each consist of two steps and load case (3) consists of a single step.

Using Multiple Nonlinear Load Cases

A multiple nonlinear load case analysis is conceptually equivalent to executing each nonlinear load case as a stand-alone analysis, with any steps comprising the common base state repeated in each stand-alone analysis. A multiple nonlinear load case analysis can be more efficient than the equivalent stand-alone analyses when the execution costs of the analysis preprocessor and the solution of the base state are significant relative to the cost of the nonlinear load cases.

If only the linear response of the structure is required for each load case, a linear multiple load case analysis may be more efficient than a nonlinear load case analysis. For more information, see *Multiple Linear Load Case Analysis*.

Defining Nonlinear Load Cases

The steps comprising a nonlinear load case are defined within an input file specific to that load case. A set of nonlinear load case input files is then referenced, along with the input file defining the model, in a manifest file. The input file defining the model must be the first input file given in the manifest file. The manifest file is submitted for execution.

The input file that defines the model can also include the steps that define the first nonlinear load case or the shared base state for all nonlinear load cases. Each additional input file given in the manifest file defines a new nonlinear load case. You can define one and only one load case within each input file.

Input File Usage: **MANIFEST*, EVOLUTION TYPE=HISTORY
name of the input file defining the model and, optionally, the base state or first

nonlinear load case
names of the input files defining each subsequent nonlinear load case

Execution of Nonlinear Load Cases

You run a multiple load case analysis by submitting the manifest file on the command line during the **abaqus** execution procedure.

Nonlinear load cases are executed in a sequential fashion in the order they are defined in the manifest file. The input file defining a given nonlinear load case is processed by the analysis input file processor before execution of the nonlinear load case. If errors are encountered during either the input file processing or during the execution of a nonlinear load case, the analysis attempts to proceed to the next nonlinear load case.

The base state and all nonlinear loads cases share the printed output (`.dat`) file, message (`.msg`) file, and status (`.sta`) file for the job. The base state and all nonlinear load cases can either share the same output database or create separate output databases (see *Results Output Format*). If each nonlinear load case creates its own output database, the base state results are included in the output database for each nonlinear load case.

Procedures

Each nonlinear load case can consist of any number of general and perturbation procedures. Rules for propagation of step attributes (such as loads, boundary conditions, and output requests) follow the same conventions as those for stand-alone analyses. For more information, see *Multiple Steps*.

Specifying the Base State

By default, the base state for all nonlinear load cases is the initial state of the model (the model plus any initial conditions). Any steps that you define along with the model in the first input file given in the manifest file form the first nonlinear load case. This is shown schematically for a sample set of input files in *Figure 1*.

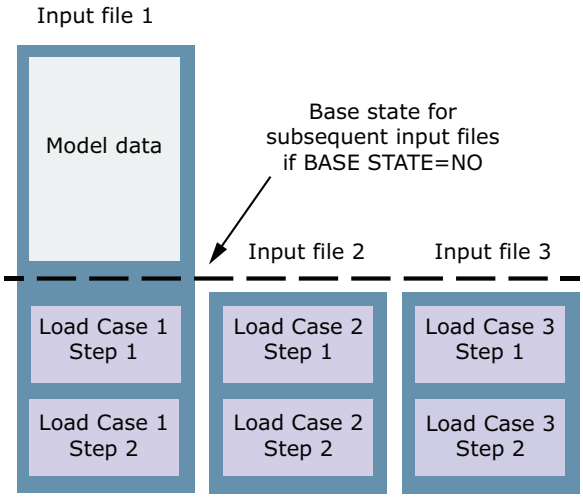


Figure 1: Impact of BASE STATE=NO on the selection of the base state for a sample multiple nonlinear load case analysis.

You can change the default behavior so that the steps that you define in the first input file given in the manifest file form the base state for all nonlinear load cases. This is shown schematically for a sample set of input files in *Figure 2*.

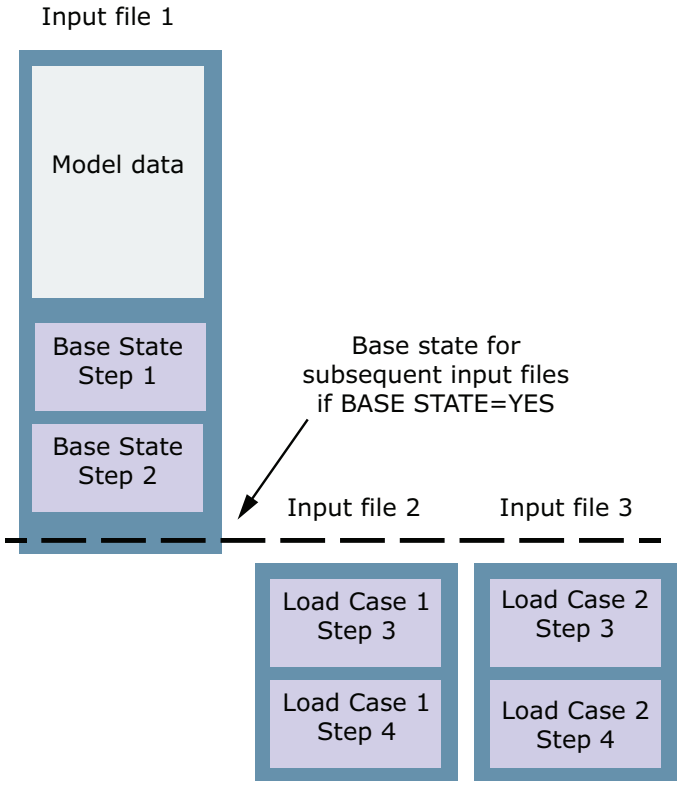


Figure 2: Impact of BASE STATE=YES on the selection of the base state for a sample multiple nonlinear load case analysis.

For example, consider the analysis of an airplane subject to three nonlinear load cases: (1) slow ascent and then bank left, (2) rapid ascent and then bank right, and (3) descent. In this example load cases (1) and (2) each consist of two steps and load case (3) consists of a single step. Initial velocities are applied to the airplane and serve as the base state for all three load cases. *Figure 3* and *Figure 4* show two possible configurations for the input files in this example.

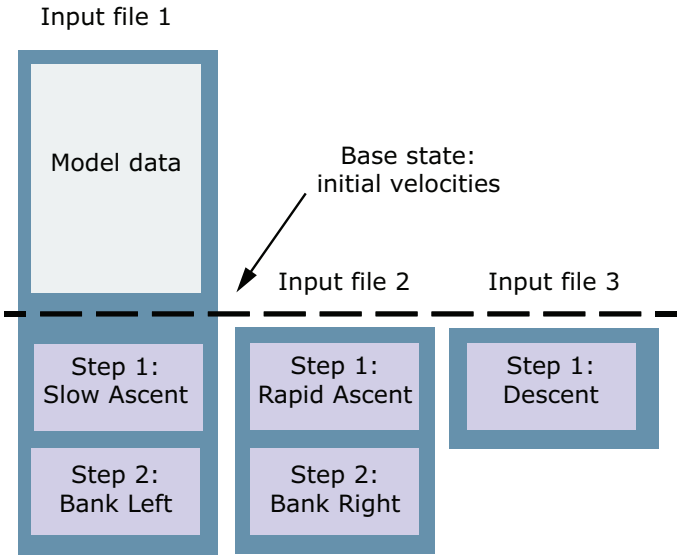


Figure 3: First example configuration.

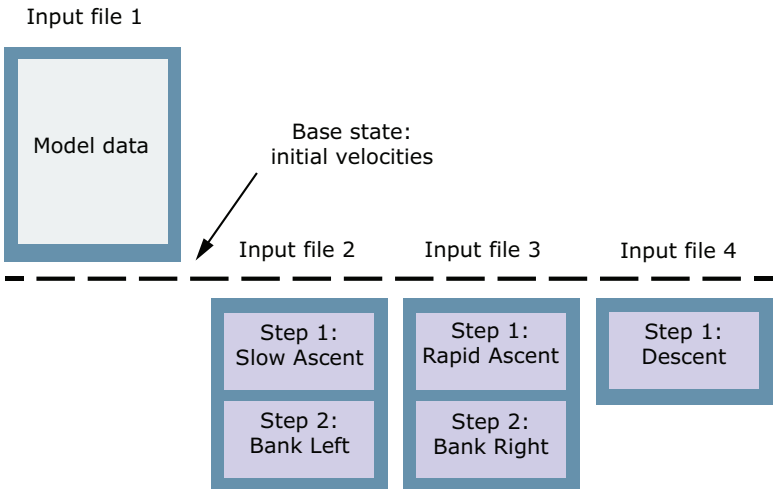


Figure 4: Second example configuration.

Input File Usage: Use the following option to use the initial state of the model as the base state:

**MANIFEST*, BASE STATE=NO (default)

Use the following option to specify that the steps of the first input file form the base state:

**MANIFEST*, BASE STATE=YES

Results Output Format

You can write results for the base state and all nonlinear load cases to the same output database. To do so, the step name must be unique for all steps in the base state and in all load cases. The output database files are named according to the analysis job name.

Alternatively, you can write results to separate output databases for each load case. In this case the base state results are duplicated in each output database. The step name must be unique for all steps across the base state and any individual load case. The first set of output database files are named according to the analysis job name. Each subsequent set of output database files is named according to the input file name for the nonlinear load case.

Input File Usage: Use the following option to write all output to a single output database:

**MANIFEST*, RESULTS=APPEND

Use the following option to write output to separate output databases:

**MANIFEST*, RESULTS=NEW

Support for Element and Contact Pair Removal

If element or contact pair removal (*Element and Contact Pair Removal and Reactivation*) can be used during some nonlinear load cases and you cannot activate the capability in the first input file of the manifest file, you must activate the capability in the manifest file.

Input File Usage: Use the following option to allow element removal during the analysis:

*MANIFEST, MODEL CHANGE

Reducing Output File Size

By default, results are written to the output databases at the start of the first step of each nonlinear load case. This can result in the duplication of a significant amount of data if the number of nonlinear load cases is large or the results requested for each nonlinear load case are identical. The size of the output databases can be reduced by skipping output at the start of each nonlinear load case.

Input File Usage: Use the following option to specify that results should not be written for the zero increment of the first step of each nonlinear load case:

*MANIFEST, ZERO INCREMENT=NO

Use the following option to write output for the zero increment of the first step of each nonlinear load case:

*MANIFEST, ZERO INCREMENT=YES (default)

Limitations

You cannot use the multiple nonlinear load case capability with the following features:

- Adaptive meshing (*ALE Adaptive Meshing and Remapping in Abaqus/Standard*)
- Parametric modeling (*Parametric Modeling*)
- Element progressive activation (*Progressive Element Activation*)
- Extended finite element method (*Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*)
- Adjoint design sensitivities (*Adjoint Design Sensitivity Analysis Overview*)
- Results transfer (*About Transferring Results between Abaqus Analyses*)

In addition, you can use the following features only in the base state:

- Application of a geostatic stress field (*Geostatic Stress State*)
- Substructure generation (*Generating Substructures*)

Input File Template

The following example shows a nonlinear load case analysis where the first input file defines the model and the steps for the first nonlinear load case. Each subsequent input file defines the steps for the next nonlinear load case.

```
*MANIFEST, EVOLUTION TYPE=HISTORY, RESULTS=NEW, BASE STATE=NO
Data line to specify the input file defining the model and the first nonlinear load case
Data line to specify the input file defining the next nonlinear load case
...
```

The following example shows a nonlinear load case analysis where the first input file defines the model and the base state for all nonlinear load cases. Each subsequent input file defines the steps for the next nonlinear load case.

```
*MANIFEST, EVOLUTION TYPE=HISTORY, RESULTS=NEW, BASE STATE=YES
Data line to specify the input file defining the model and the base state steps
Data line to specify the input file defining the first nonlinear load case
```

Data line to specify the input file defining the next nonlinear load case
...

Step Cycling

Products: Abaqus/Standard

References:

- **STEP CYCLING CONTROL*

Overview

Step cycling:

- is used to repeat a step or series of steps over a set number of cycles;
- can be used in conjunction with steps that are not defined as part of a step cycling sequence;
- allows control of output at a specified cycle frequency; and
- can be more efficient than explicitly defining the equivalent steps individually.

Introduction

For some analyses it is desirable to repeat a step or series of steps over a large number of cycles. Common examples include battery aging, wear, erosion, and material fatigue analyses. Explicitly redefining the same steps to simulate a large number of cycles is cumbersome and inefficient. The step cycling capability provides a more convenient and efficient means to define cycling behavior.

Defining a Step Cycling Sequence

You can define a step or a series of steps to be repeated within a step cycling sequence. Abaqus/Standard repeats the steps according to the number of cycles you specify as part of the step cycling sequence definition. Steps outside of a step cycling sequence definition are executed only once. If you begin a step cycling sequence definition but do not specify an end to the sequence, Abaqus/Standard cycles all steps following the start of the cycling sequence.

You can assign a name to a step cycling sequence. Abaqus/Standard prepends this name to the step name written to the output databases for steps within the step cycling sequence.

Input File Usage: Use the following option to begin the step cycling definition:

**STEP CYCLING*, START, NAME=*name*
number of cycles

Use the following option to end the step cycling definition:

**STEP CYCLING*, END

Controlling or Terminating a Step Cycling Sequence

If you use the step controls mechanism (*Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) to end an intermediate step of a step cycling sequence, Abaqus/Standard will proceed to the next step in the step cycle definition. Some applications might also require terminating a cycling sequence. You can use the step cycling control mechanism to end a cycling sequence before the specified number of cycles is

reached. It is recommended that you end a cycling sequence during the last step of the sequence. However, if your application requires ending a cycling sequence in any intermediate step within the cycle and there are other steps following the cycling sequence, your model definitions must ensure that any subsequent changes to loads and boundary conditions are meaningful and correct.

Input File Usage: Use the following option to end a cycling sequence before the specified number of cycles is reached:

**STEP CYCLING CONTROL*, NAME=*name*, ACTION=END CYCLING

Reducing Output File Size

By default, Abaqus/Standard writes results to the output databases for all cycles within a step cycling sequence. This can result in a significant amount of data if the number of cycles is large. You can reduce the size of the output databases by restricting output to a specific cycle frequency as part of the step cycling sequence definition. Abaqus/Standard always writes output in the last cycle of a step cycling sequence.

Input File Usage: Use the following option to specify the output frequency:

**STEP CYCLING*, START, NAME=*name*
number of cycles, frequency at which to write output

Results Output Format

By default, Abaqus/Standard writes all results for a step cycling sequence to the same output database. However, for analyses consisting of a series of steps not part of a cycling sequence followed by only a single step cycling sequence, Abaqus/Standard can distribute the results to multiple output files. You can specify a cycle frequency at which Abaqus/Standard directs output to new output databases.

The names of the new output databases will include an integer suffix indicating the index of the last cycle written to the file. For example, if an analysis consists of 1000 cycles and you specify a cycle frequency of 200, Abaqus/Standard writes six output databases. The name of the first output database is `jobName_1.odb`, and it contains the first cycle only. The names of the subsequent output databases are of the form `jobName_`*N*`.odb`, where *N* is 200, 400, 600, 800, and 1000.

Input File Usage: Use the following option to distribute output to multiple output databases:

**STEP CYCLING*, START, NAME=*name*
number of cycles, frequency at which to write output, frequency at which to create a new output database

Continuation and Results Import from a Step Cycling Analysis

To optimize performance, Abaqus/Standard stores only the final state of the steps comprising a step cycling sequence on the analysis database. Therefore, subsequent continuation analyses (*Analysis Continuation Techniques*) should use the step id of the continuation step without regard to a specific cycle. For example, if you perform a restart from an analysis consisting of two steps repeated through *N* cycles, in a subsequent restart analysis you would specify either Step 1 or Step 2 as the step from which to continue the analysis.

In contrast, Abaqus/Standard writes results to the output databases for each cycle of a step cycling sequence, in accordance with the user's output requests. Therefore, step ids on the output databases are sequential across all steps of the analysis. If you import results into a subsequent analysis, such as for initial conditions (*Initial Conditions*) or external fields (*General Capability for Importing External Fields*), you should use the step id

consistent with the step ids present on the output databases. For example, if an analysis consists of two steps repeated through 50 cycles, the step ids available for importing results would lie in the range 1 to 100.

Limitations

Step cycling is subject to the following limitations:

- Perturbation procedures cannot be included in a step cycling sequence.
- Restart can be performed only from the last cycle of a step cycling sequence (*Restarting an Analysis*).
- Import cannot be performed between two analyses if either analysis includes a step cycling sequence (*About Transferring Results between Abaqus Analyses*).

In addition, a step cycling sequence cannot include the following features:

- Adaptive meshing (*ALE Adaptive Meshing and Remapping in Abaqus/Standard*)
- Adjoint design sensitivities (*Adjoint Design Sensitivity Analysis Overview*)
- Parametric modeling (*Parametric Modeling*)
- Element progressive activation (*Progressive Element Activation*)
- Extended finite element method (*Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*)
- Multiple nonlinear load cases (*Multiple Nonlinear Load Case Analysis*)

Input File Template

```
*STEP CYCLING, START, NAME=cycname
number of cycles, frequency at which to write output, frequency at which to create a new output database
*STEP
Input defining the first step of the step cycling sequence
*END STEP
Repeat as necessary for additional steps within the same step cycling sequence
*STEP CYCLING, END
```

Direct Linear Equation Solver

Products: Abaqus/Standard Abaqus/CAE

References:

- *Abaqus/Standard and Abaqus/Explicit Execution*
- *Environment File Settings*
- *Iterative Linear Equation Solver*
- *Parallel Execution in Abaqus/Standard*
- *Configuring analysis procedure settings*

Overview

Linear equation solution is used in linear and nonlinear analysis. In nonlinear analysis Abaqus/Standard uses the Newton method or a variant of it, such as the Riks method, within which it is necessary to solve a set of linear equations at each iteration. The direct linear equation solver finds the exact solution to this system of linear equations (up to machine precision). The direct linear equation solver in Abaqus/Standard:

- uses a sparse, direct, Gauss elimination method; and
- often represents the most time consuming part of the analysis (especially for large models)—the storage of the equations occupies the largest part of the disk space during the calculations.

The Sparse Solver

The direct sparse solver uses a "multifront" technique that can reduce the computational time to solve the equations dramatically if the equation system has a sparse structure. Such a matrix structure typically arises when the physical model is made from several parts or branches that are connected together; a spoked wheel is a good example of a structure that has a sparse stiffness matrix. Space frames and other structures modeled with beams, trusses, and shells often have sparse stiffness matrices. In contrast, a *blocky* structure—such as a single, solid, three-dimensional block (see *Elastic-plastic line spring modeling of a finite length cylinder with a part-through axial flaw*)—provides little opportunity for the sparse solver to reduce the computer time. For large blocky structures, the iterative linear equation solver may be more efficient (see *Iterative Linear Equation Solver*).

Input File Usage: Use the following option to use the default direct sparse solver:

**STEP*

Abaqus/CAE Usage: Step module: step editor: **Other**: **Method: Direct**

Setting Controls for the Direct Linear Solver

The linear equation solver can optimize elimination of constraint equations associated with hard contact and hybrid elements. There are two potential undesirable side-effects associated with this option:

- Possible small degradation of solution accuracy may adversely impact the nonlinear convergence behavior.
- Possible minor performance degradation for models without hard contact constraints and/or hybrid elements.

Input File Usage: Use the following option to turn on constraint optimization:

**SOLVER CONTROLS*, CONSTRAINT OPTIMIZATION

Abaqus/CAE Usage: You cannot specify constraint optimization in Abaqus/CAE.

Iterative Linear Equation Solver

Products: Abaqus/Standard Abaqus/CAE

References:

- **STEP*
- **SOLVER CONTROLS*
- *Parallel Execution in Abaqus/Standard*
- *Customizing solver controls*

Overview

The iterative linear equation solver in Abaqus/Standard:

- can be used for linear and nonlinear static, quasi-static, heat transfer, geostatic, and coupled pore fluid diffusion and stress analysis solution procedures;
- should be used only for large, well-conditioned models for which the direct sparse solver (see *Direct Linear Equation Solver*) requires a prohibitively large number of floating point operations;
- is likely to be dramatically faster than the direct equation solver for large, well-conditioned, blocky structures;
- runs totally in-core and uses less storage than the direct sparse solver (memory and disk combined);
- can be used only with three-dimensional models;
- must be the only solver invoked in the analysis (that is, you cannot use the iterative solver in one step and the direct solver in another);
- cannot be used with automatic stabilization with an adaptive damping factor (see *Adaptive Automatic Stabilization Scheme*); and
- can be used with a constant damping factor if stabilization is necessary (see *Automatic Stabilization of Static Problems with a Constant Damping Factor*).

Iterative Solver Basics

The iterative solver in Abaqus/Standard can be used to find the solution to a linear system of equations and can be invoked in a linear or nonlinear static, quasi-static, geostatic, pore fluid diffusion, or heat transfer analysis step. Since the technique is iterative, a converged solution to a given system of linear equations cannot be guaranteed. In cases where the iterative solver fails to converge to a solution, modifications to the model might be necessary to improve the convergence behavior. In some cases the only choice might be to use the direct solver to obtain a solution. When the iterative solver converges, the accuracy of this solution depends on the relative tolerance that is used; the default tolerance is sufficiently accurate for most purposes. However, tolerance adjustments for particular analyses might improve the overall performance of the simulation. In addition, the performance of the iterative solver relative to the direct sparse solver is highly sensitive to the model geometry, favoring blocky type structures (that is, models that look more like a cube than a plate) with a high degree of mesh connectivity and a relatively low degree of sparsity. These types of models often demand the most computational and storage resources for the direct sparse solver. Models with a lesser degree of connectivity (often said to have a higher degree of sparsity), such as thin, shell-like structures, are much more suited to the direct sparse solver (see *Direct Linear Equation Solver*).

Input File Usage: Use the following option to invoke the iterative solver:

*STEP, SOLVER=ITERATIVE

Abaqus/CAE Usage: Step module: step editor: **Other**: **Method: Iterative**

Iterative Solution Technique

The iterative solution technique in Abaqus/Standard is based on Krylov methods employing a preconditioner. This solver uses the following general strategy:

1. The Krylov method solver iterates on the system of equations generated by the finite element method while a preconditioner is applied at each iteration.
2. The preconditioner is calculated only once at the beginning of each linear system solve and is used to accelerate the convergence of the Krylov method.
3. In parallel, all components of the iterative solution process (including matrix assembly, preconditioner setup, and the actual solve using the Krylov method) are handled in a hybrid manner using an MPI and thread-based parallel implementation.

The process outlined above is performed entirely internal to Abaqus/Standard, with no user intervention required.

Convergence of the Linear System of Equations

To generate the solution to the system of linear algebraic equations (denoted by the matrix equation $\boldsymbol{Ku = f}$, where K is the global stiffness matrix, f is the load vector, and u is the desired displacement solution), a sequence of Krylov solver iterations is performed, where an approximate solution gets closer to the exact solution at each iteration. The error in the approximate solution is measured by the relative residual of the linear system, defined by $||\boldsymbol{Ku - f}|| \,/\, ||\boldsymbol{f}||$, where $||\cdot||$ is the $\boldsymbol{L^2}$ norm. The term "convergence" is used to describe this process, and the approximate solution is said to be converged when the relative residual is below a specified tolerance. By default, this tolerance is 10^{-6} for both linear perturbation and general nonlinear procedures. The linear solver convergence tolerance is independent from the nonlinear convergence process (that is, Newton-Raphson method) tolerances that are used to determine if analysis increments converge. The latter are the same regardless of the choice of linear equation solver (iterative or direct).

The rate at which the approximate solution converges is directly related to the conditioning of the original system of equations. A linear system that is well conditioned will converge faster than an ill-conditioned system. If the residual does not converge to tolerance within the maximum number of iterations, the iterative solver is said to have encountered a non-convergence and Abaqus/Standard issues a warning message. However, the analysis will continue running and in some cases the Newton-Raphson iterations within increments might continue to converge.

Setting Controls for the Iterative Linear Solver

The default controls provided in Abaqus/Standard are usually sufficient. However, a method for overriding the default relative convergence tolerance and maximum number of solver iterations is provided.

Resetting the Solver Controls

You can specify that the solver controls be reset to their default values.

Input File Usage: *SOLVER CONTROLS, RESET

Abaqus/CAE Usage: Step module: **Other->Solver Controls->Edit**: **Reset all parameters to their system-defined defaults**

Specifying the Relative Convergence Tolerance

By default, this tolerance is 10^{-6} for all linear perturbation and general nonlinear procedures. For nonlinear problems the accuracy of the linear solution can impact the convergence of the Newton method. In some cases it might be necessary to manually specify the iterative solver relative tolerance to improve the convergence of the Newton-Raphson method or to improve performance.

Input File Usage: **SOLVER CONTROLS*
relative tolerance for convergence

Abaqus/CAE Usage: Step module: **Other->Solver Controls->Edit**: **Specify**: **Relative tolerance: Specify:** *relative tolerance for convergence*

Specifying the Maximum Number of Solver Iterations

In rare instances the linear solver might require more than the default number of iterations to converge to the desired level of accuracy. In this case you can increase the maximum number of iterations allowed by the iterative solver (the default value is 500).

Input File Usage: **SOLVER CONTROLS*
, max number of solver iterations

Abaqus/CAE Usage: Step module: **Other->Solver Controls->Edit**: **Specify**: **Max. number of iterations: Specify:** *max number of solver iterations*

Deciding to Use the Iterative Solver

Many factors must be carefully weighed before deciding to use the iterative solver in Abaqus/Standard, such as element type, contact and constraint equations, material and geometric nonlinearities, and material properties, all of which can impact robustness and performance. In cases where the model is ill conditioned, the iterative solver might converge very slowly or fail to converge. For example, this can occur if many elements have poor aspect ratios.

In general, the iterative solver might not converge for models for which the direct solver issues the following:

- Negative eigenvalue warning messages
- Zero pivot warning messages
- Numerical singularity warning messages

These warning messages indicate that the system matrix is very ill conditioned, and iterative solver convergence is not guaranteed for an ill-conditioned system. For example, the direct solver issues one of these warning messages for an underconstrained system, and the iterative solver might fail during the preconditioner setup phase or might not converge within the maximum number of iterations for this system.

In addition to these general limitations of the iterative solver, the iterative solver in Abaqus/Standard has the following limitations:

- Inertia relief loads are not supported.
- Geostatic analyses and pore fluid diffusion analyses are not supported for models with constraints using Lagrange multipliers.
- Fully coupled thermal-stress and thermal-electrical analyses are not supported.
- Steady-state transport analysis is not supported.

Apart from the robustness issues (relating mainly to the rate of convergence and limitations), the iterative solver is more likely to outperform the direct sparse solver for blocky models that require a very large number of floating point operations for factorization. Typically, the number of degrees of freedom in the model must be greater than one million before the iterative solver is comparable to the direct solver in terms of run time.

Element Type and Model Geometry

The most basic modeling issue that will affect the performance of the iterative solver is the model geometry, which must be carefully considered when deciding if the iterative solver is suited for a particular model. In general, models that are blocky in nature (that is, look more like a cube than a plate) and are dominated by solid elements will behave well with the iterative solver. Although structural elements such as beams and shells are supported, models with structural elements will not perform optimally; the direct sparse solver should be used instead for such models. Common modeling techniques such as coating solid elements with a thin layer of membrane elements to recover accurate stresses on the boundary or fixing rigid body motion with weak springs might not work with the iterative solver. Applying loads or boundary conditions to large node sets using locally transformed coordinate systems can also cause convergence difficulties. All of these techniques are likely to lead to extremely slow convergence or stagnation.

Another factor that can influence the convergence of the iterative solver is the quality of the elements. Blocky models, such as an engine block, that contain many poorly shaped elements with high aspect ratios can also lead to poor iterative solver convergence. It is a good idea to look for warning messages about poorly shaped elements when evaluating the performance of the iterative solver.

Using cohesive elements with the iterative solver will likely lead to nonconvergence.

Contact

Since contact is a form of nonlinear analysis, special care must be taken in selecting the convergence tolerance for the iterative solver (see *Nonlinear Analysis* below). Therefore, it is recommended to run the model through a static perturbation analysis before proceeding to the nonlinear problem. This will demonstrate how the iterative solver will perform for the specific model geometry without the added difficulty of nonlinear convergence.

Contact constraints enforced with Lagrange multipliers (that is, direct enforcement) or very high penality stiffness might cause the iterative solver to fail to converge.

The iterative solver does not support pore fluid contact, regardless of the contact formulation used.

Material Properties

When deciding to use the iterative solver, the variation of material properties in the model should be considered. Models that have very large discontinuities in material behavior (many orders of magnitude) will most likely converge slowly and possibly stagnate. Models with almost fully incompressible material (that is, a Poisson ratio close to 0.5) will converge very slowly.

Nonlinear Analysis

The iterative solver can be used to solve the linear system of algebraic equations that arises at each iteration of the Newton procedure. However, the convergence of the nonlinear problem will be affected by the convergence of the iterative linear solver. The actual impact depends on the particular model and type of nonlinearities present. In most cases the default iterative solver tolerance of 10^{-6} is sufficient to maintain the convergence of the Newton method; however, in some cases a smaller linear solver tolerance (for example, 10^{-9}) can be used to improve the nonlinear convergence.

If a nonlinear analysis that uses the iterative solver fails to converge, it is often difficult to determine if this is due to the approximate linear equation solution of the iterative solver or if the Newton process itself is failing to converge. If nonlinear convergence problems occur, the direct solver can be used—given the problem is

solvable using the direct solver due to solution cost—to eliminate the approximate linear solution as a possible source of the problem.

Static Stress/Displacement Analysis

In this section:

- *About Static Stress Analysis Procedures*
- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Direct Cyclic Analysis*
- *Eigenvalue Buckling Prediction*
- *Unstable Collapse and Postbuckling Analysis*
- *Linear Elastic Fatigue Crack Growth Analysis*
- *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*

About Static Stress Analysis Procedures

Overview

A static stress procedure is one in which inertia effects are neglected.

Several static stress analysis procedures are available in Abaqus/Standard:

General analysis

- *Static Stress Analysis* is used for stable problems and can include linear or nonlinear response.
- *Quasi-Static Analysis* is used to analyze the transient response of structures considering time-dependent material behavior (creep and swelling, viscoelasticity, and viscoplasticity). A quasi-static analysis can be linear or nonlinear.
- *Direct Cyclic Analysis* is used to calculate the stabilized cyclic response of the structure directly. It uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized cyclic solution iteratively.

Buckling analysis

- *Eigenvalue Buckling Prediction* is used to estimate the critical (bifurcation) load of “stiff” structures. It is a linear perturbation procedure.
- *Unstable Collapse and Postbuckling Analysis* is used to estimate the unstable, geometrically nonlinear collapse of a structure. The method can also be helpful in obtaining a solution in other types of unstable problems, and it is often suitable for limit load analyses.

Fatigue analysis

- *Linear Elastic Fatigue Crack Growth Analysis* is used to predict delamination/debonding growth at the brittle material interfaces in laminated composites and to predict crack growth in bulk brittle materials under cyclic loading.
- *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach* is used to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding growth at the interfaces in laminated composites based on the direct cyclic approach in conjunction with the damage extrapolation technique.

References:

Static Stress Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A static stress analysis:

- is used when inertia effects can be neglected;
- can be linear or nonlinear;
- ignores time-dependent material effects (creep, swelling, viscoelasticity) but takes hysteretic behavior for hyperelastic materials into account; and
- takes into account rate-dependent yield and friction by default, but you can optionally ignore them.

References:

- *Defining an Analysis*
- *About Static Stress Analysis Procedures*
- **STATIC*
- *Configuring a static, general procedure*

Time Period

During a static step you assign a time period to the analysis. This is necessary for cross-references to the amplitude options, which can be used to determine the variation of loads and other externally prescribed parameters during a step (see *Amplitude Curves*). In some cases this time scale is quite real—for example, the response might be caused by temperatures varying with time based on a previous transient heat transfer run; or the material response might be rate dependent (rate-dependent plasticity), so that a natural time scale exists. Other cases do not have such a natural time scale; for example, when a vessel is pressurized up to limit load with rate-independent material response. If you do not specify a time period, Abaqus/Standard defaults to a time period in which "time" varies from 0.0 to 1.0 over the step. The "time" increments are then fractions of the total period of the step.

Linear Static Analysis

Linear static analysis involves the specification of load cases and appropriate boundary conditions. If all or part of a problem has linear response, substructuring is a powerful capability for reducing the computational cost of large analyses (see *Using Substructures*).

Nonlinear Static Analysis

Nonlinearities can arise from large-displacement effects, material nonlinearity, and/or boundary nonlinearities such as contact and friction (see *General and Perturbation Procedures*) and must be accounted for. If geometrically nonlinear behavior is expected in a step, the large-displacement formulation should be used. In most nonlinear analyses the loading variations over the step follow a prescribed history such as a temperature transient or a prescribed displacement.

Input File Usage: Use the following option to specify that a large-displacement formulation is used for a static step:

*STEP, NLGEOM

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Basic**: **Nlgeom**: **On** (to activate the large-displacement formulation)

Unstable Problems

Some static problems can be naturally unstable, for a variety of reasons.

Buckling or Collapse

In some geometrically nonlinear analyses, buckling or collapse might occur. In these cases a quasi-static solution can be obtained only if the magnitude of the load does not follow a prescribed history; it must be part of the solution. When the loading can be considered proportional (the loading over the complete structure can be scaled with a single parameter), a special approach—called the "modified Riks method"—can be used, as described in *Unstable Collapse and Postbuckling Analysis*.

Input File Usage: *STATIC, RIKS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**

Local Instabilities

In other unstable analyses the instabilities are local (for example, surface wrinkling, material instability, or local buckling), in which case global load control methods such as the Riks method are not appropriate. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Incrementation

Abaqus/Standard uses Newton's method to solve the nonlinear equilibrium equations. Many problems involve history-dependent response; therefore, the solution usually is obtained as a series of increments, with iterations to obtain equilibrium within each increment. Increments must sometimes be kept small (in the sense that rotation and strain increments must be small) to ensure correct modeling of history-dependent effects. Most commonly the choice of increment size is a matter of computational efficiency: if the increments are too large, more iterations are required. Furthermore, Newton's method has a finite radius of convergence; too large an increment can prevent any solution from being obtained because the initial state is too far away from the equilibrium state that is being sought—it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic Incrementation

In most cases the default automatic incrementation scheme is preferred because it selects increment sizes based on computational efficiency.

Input File Usage: *STATIC

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Incrementation**: **Type**: **Automatic** (default)

Direct Incrementation

Direct user control of the increment size is also provided because if you have considerable experience with a particular problem, you might be able to select a more economical approach.

Input File Usage: **STATIC*, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Incrementation**: **Type**: **Fixed**

Special Cases for Direct Incrementation

With direct user control, the solution to an increment can be accepted after the maximum number of iterations allowed has been completed (as defined in *Commonly Used Control Parameters*), even if the equilibrium tolerances are not satisfied. This approach is not recommended; it should be used only in special cases when you have a thorough understanding of how to interpret results obtained in this way. Very small increments and a minimum of two iterations are usually necessary if this option is used.

Input File Usage: **STATIC*, DIRECT=NO STOP

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Other**: **Accept solution after reaching maximum number of iterations**

Steady-State Frictional Sliding

In a static analysis procedure you can model steady-state frictional sliding between two deformable bodies or between a deformable and a rigid body that are moving with different velocities by specifying the motions of the bodies as predefined fields. In this case it is assumed that the slip velocity follows from the difference in the user-specified velocities and is independent of the nodal displacements, as described in *Coulomb friction*.

Since this frictional behavior is different from the frictional behavior used without steady-state frictional sliding, discontinuities might arise in the solutions between an analysis step in which relative velocity is determined from predefined motions and prior steps. An example is the discontinuity that occurs between the initial preloading of the disc pads in a disc brake system and the subsequent braking analysis where the disc spins with a prescribed rotation. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to the analysis step in which predefined motion is specified use a zero coefficient of friction. You can then modify the friction properties in the steady-state analysis to use the desired friction coefficient (see *Changing Friction Properties during an Abaqus/Standard Analysis*).

Input File Usage: **MOTION*

Abaqus/CAE Usage: Predefined motion fields are not supported in Abaqus/CAE.

Rate-Dependent Yield and Friction

Rate-dependent yield (*Rate-Dependent Yield*) is active during a static analysis. You can control whether to consider or ignore the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient within the step.

Input File Usage: Use the following option to consider rate dependence within the step (default):

**STATIC*, RATE DEPENDENCE=ON

Use the following option to ignore rate dependence within the step:

**STATIC*, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified. *Initial Conditions* describes all the available initial conditions.

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6); to warping degree of freedom 7 in open-section beam elements; or, if hydrostatic fluid elements are included in the model, to fluid pressure degree of freedom 8. If boundary conditions are applied to rotation degrees of freedom, you must understand how finite rotations are handled by Abaqus (see *Boundary Conditions*). During the analysis prescribed boundary conditions can be varied using an amplitude definition (see *Amplitude Curves*).

Loads

The following loads can be prescribed in a static stress analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Predefined Fields

The following predefined fields can be specified in a static stress analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in a static stress analysis, you can specify nodal temperatures as a predefined field. Any difference between the applied and initial temperatures causes thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.
- Although pore fluid pressure is not a degree of freedom in a static stress analysis, you can specify nodal pore fluid pressure as a predefined field variable (see *Pore Fluid Pressure*). The total stress is computed as the sum of the effective stress and the pore fluid pressure. When the solid grains are compressible (finite bulk modulus), the effective strain is computed as the total strain minus the volumetric strain caused by the pore pressure acting on the solid grain.

Material Options

Most material models that describe mechanical behavior are available for use in a static stress analysis. The following material properties are not active during a static stress analysis: acoustic properties, thermal properties

(except for thermal expansion), mass diffusion properties, electrical conductivity properties, and pore fluid flow properties.

Rate-dependent yield (*Rate-Dependent Yield*, hysteresis (*Hysteresis in Elastomers*), and two-layer viscoplasticity (*Two-Layer Viscoplasticity*) are the only time-dependent material responses that are active during a static analysis. An appropriate time scale must be specified so that Abaqus/Standard can treat the rate dependence of the material responses correctly.

Static creep and swelling problems and time-domain viscoelastic models are analyzed by the quasi-static procedure (*Quasi-Static Analysis*). When you use any of these time-dependent material models in a static analysis, Abaqus/Standard obtains a rate-independent elastic solution and the chosen time scale does not affect the material response. For creep and swelling behavior this indicates that the loading is applied instantaneously compared with the natural time scale over which creep effects take place.

The same concept of instantaneous load application applies to time-domain viscoelastic behavior. You can also obtain the fully relaxed long-term viscoelastic solution directly in a static procedure without having to perform a transient analysis; this choice is meaningful only when you define time-domain viscoelastic material properties. If you request the long-term viscoelastic solution, Abaqus/Standard ramps the internal stresses associated with each of the Prony series terms gradually from their values at the beginning of the step to their long-term values at the end of the step.

For the two-layer viscoplastic material model, you can obtain the long-term response of the elastic-plastic network alone.

When frequency-domain viscoelastic material properties are defined (see *Frequency Domain Viscoelasticity*), you must specify the corresponding elastic moduli as long-term elastic moduli. This implies that the response corresponds to the long-term elastic solution, regardless of the time period specified for the step.

Input File Usage: Use the following option to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity:

**STATIC*, LONG TERM

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General** or **Static, Riks**: **Other**: **Obtain long-term solution with time-domain material properties**

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a static stress analysis (see *Choosing the Appropriate Element for an Analysis Type*). Although velocities are not available in a static stress analysis, dashpots can still be used (they can be useful in stabilizing an unstable problem). The relative velocity is calculated as described in *Dashpots*.

Acoustic elements are not active in a static step. Consequently, if an acoustic-solid analysis includes a static step, only the solid elements deform. If the deformations are large, the acoustic and solid meshes might not conform, and subsequent acoustic-structural analysis steps might produce misleading results. See *About ALE Adaptive Meshing* for information on using the adaptive meshing technique to deform the acoustic mesh.

Output

The element output available for a static stress analysis includes stress; strain; energies; the values of state, field, and user-defined variables; and composite failure measures. The nodal output available includes displacements, reaction forces, and coordinates. All the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,NLGEOM)
Once NLGEOM is specified, it will be active in all subsequent steps
*STATIC, DIRECT
Data line to define direct time incrementation
*BOUNDARY
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*END STEP
**
*STEP
*STATIC
Data line to control automatic time incrementation
*BOUNDARY, OP=MOD
Data lines to modify or add zero-valued or nonzero boundary conditions
*CLOAD, OP=NEW
Data lines to specify new concentrated loads; all previous concentrated
loads will be removed
*DLOAD, OP=MOD
Data lines to specify additional or modified distributed loads
*TEMPERATURE and/or *FIELD
Data lines to specify additional or modified values of predefined fields
*END STEP
```

Quasi-Static Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A quasi-static stress analysis in Abaqus/Standard:

- is used to analyze problems with time-dependent material response (creep, swelling, viscoelasticity, and two-layer viscoplasticity);
- is used when inertia effects can be neglected; and
- can be linear or nonlinear.

See *Mass Scaling* and *Explicit Dynamic Analysis* for information on conducting quasi-static analysis in Abaqus/Explicit. See *Implicit Dynamic Analysis Using Direct Integration* for information on conducting quasi-static analysis using a dynamic procedure in Abaqus/Standard.

References:

- *Defining an Analysis*
- *About Static Stress Analysis Procedures*
- **VISCO*
- *Configuring a transient, static, stress/displacement analysis with time-dependent material response*

Incrementation

You can control the time incrementation in a quasi-static analysis directly, or it can be controlled automatically by Abaqus/Standard. Automatic incrementation is preferred in almost all cases.

Fixed Incrementation

If you specify the time increments in a quasi-static analysis directly, fixed time increments equal to the specified initial time increment will be used throughout the analysis, except when the explicit creep integration scheme is used. In this case Abaqus/Standard might decrease the time increment if the stability limit is exceeded.

Input File Usage: **VISCO*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**

Automatic Incrementation

If you select automatic incrementation, the size of the time increment is limited by the accuracy of the integration. The user-specified accuracy tolerance parameter limits the maximum inelastic strain rate change allowed over an increment:

$$tolerance \geq \left(\dot{\bar{\varepsilon}}^{cr} \big|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr} \big|_{t} \right) \Delta t,$$

where t is the time at the beginning of the increment, Δt is the time increment (so that $t + \Delta t$ is the time at the end of the increment), and $\dot{\bar{\varepsilon}}^{cr}$ is the equivalent creep strain rate. To achieve accuracy, the value chosen for the accuracy tolerance parameter should be on the order of σ_{err}/E for creep problems, where σ_{err} is an acceptable level of error in the stress and E is a typical elastic modulus, or on the order of the elastic strains for viscoelasticity problems.

Input File Usage: **VISCO*, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**: **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance**: *tolerance*

Selecting Explicit Creep Integration

Nonlinear creep problems (*Rate-Dependent Plasticity: Creep and Swelling*) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For creep at very low stress levels, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases Abaqus/Standard will invoke the implicit integration scheme automatically.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine *CREEP*; you can restrict Abaqus/Standard to using this method for creep problems (with or without geometric nonlinearity included). See *Rate-Dependent Plasticity: Creep and Swelling* for further details.

Input File Usage: **VISCO*, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**: **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance**: *tolerance* and **Creep/swelling/viscoelastic integration: Explicit**

Integration Scheme for Viscoelasticity and Rate-Dependent Yield

Problems including *Time Domain Viscoelasticity* are always integrated with an unconditionally stable operator. The time step in these problems is limited only by the accuracy tolerance parameter defined above.

Problems including *Rate-Dependent Yield* and *Parallel Rheological Framework* are always integrated using an implicit, unconditionally stable method. The accuracy tolerance parameter does not limit the inelastic strain rate change and can be set equal to any nonzero value to activate automatic time incrementation.

Unstable Problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers the ability to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem

is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified, as described in *Initial Conditions*.

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6); to warping degree of freedom 7 in open-section beam elements; or, if hydrostatic fluid elements are included in the model, to fluid pressure degree of freedom 8. If boundary conditions are applied to rotation degrees of freedom, you must understand how Abaqus handles finite rotations. See *Boundary Conditions*.

Loads

The following types of loading can be prescribed in a quasi-static analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Predefined Fields

The following predefined fields can be specified in a quasi-static analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in quasi-static analysis, nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material Options

The quasi-static procedure in Abaqus/Standard is generally used to analyze quasi-static creep and swelling problems, which occur over fairly long time periods (*Rate-Dependent Plasticity: Creep and Swelling*). This procedure can also be used to analyze viscoelastic materials (*Time Domain Viscoelasticity* and *Parallel Rheological Framework*) and two-layer viscoplastic materials (*Two-Layer Viscoplasticity*). In addition, all material models that are valid in a static analysis procedure can be used.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in a quasi-static stress analysis—see *Choosing the Appropriate Element for an Analysis Type*.

Output

In addition to the usual output variables available in Abaqus/Standard (see *Abaqus/Standard Output Variable Identifiers*), the following variables are provided specifically for creep problems:

Element integration point variables:

CEEQ
Equivalent creep strain, $\int_0^t \dot{\bar{\varepsilon}}^{cr} dt$.

CESW
Magnitude of the swelling strain.

CEMAG
Magnitude of the creep strain, $\sqrt{\frac{2}{3}\boldsymbol{\varepsilon}^{cr} : \boldsymbol{\varepsilon}^{cr}}$.

CEP
Principal creep strains.

CE
Output of all of the creep strain components and CEEQ, CESW, and CEMAG.

Input File Template

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,NLGEOM)
*VISCO, CETOL=tolerance
Data line to define time incrementation and a "real" time scale
*BOUNDARY
Data lines to describe nonzero boundary conditions
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
Data lines to specify loading
*END STEP
```

Direct Cyclic Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A direct cyclic analysis:

- is a quasi-static analysis;
- uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized cyclic response of the structure iteratively;
- avoids the considerable numerical expense associated with a transient analysis;
- is ideally suited for very large problems in which many load cycles must be applied to obtain the stabilized response if transient analysis is performed;
- can be performed with linear or nonlinear material with localized plastic deformation;
- can be used to predict the likelihood of plastic ratcheting;
- assumes geometrically linear behavior and fixed contact conditions;
- uses the elastic stiffness, so the equation system is inverted only once; and
- can also be used to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding growth at the interfaces in laminated composites in a low-cycle fatigue analysis.

References:

- *Defining an Analysis*
- **DIRECT CYCLIC*
- **TIME POINTS*
- **CONTROLS*
- *Configuring a direct cyclic procedure*

Introduction

It is well known that after a number of repetitive loading cycles, the response of an elastic-plastic structure, such as an automobile exhaust manifold subjected to large temperature fluctuations and clamping loads, may lead to a stabilized state in which the stress-strain relationship in each successive cycle is the same as in the previous one. The classical approach to obtain the response of such a structure is to apply the periodic loading repetitively to the structure until a stabilized state is obtained. This approach can be quite expensive, since it may require the application of many loading cycles before the stabilized response is obtained. To avoid the considerable numerical expense associated with a transient analysis, a direct cyclic analysis can be used to calculate the cyclic response of the structure directly. The basis of this method is to construct a displacement function $\bar{u}(t)$ that describes the response of the structure at all times t during a load cycle with period T as shown in *Figure 1*.

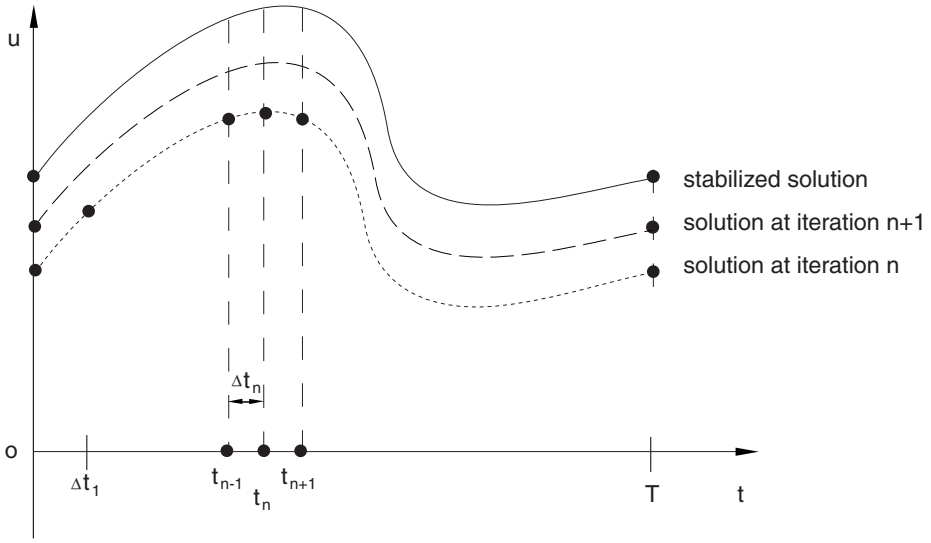


Figure 1: A displacement function at all times t during a load cycle with period T at different iterations.

A truncated Fourier series is used for this purpose,

$$\overline{u}\,(t) \;= u_0 + \sum_{k=1}^{n} \left[u_k^s \sin k\omega t + u_k^c \cos k\omega t \right],$$

where n stands for the number of terms in the Fourier series, $\omega = 2\pi/T$ is the angular frequency, and u_0, u_k^s, and u_k^c are unknown displacement coefficients associated with each degree of freedom in the problem. Abaqus/Standard solves for the unknown displacement coefficients by using a modified Newton method, with the elastic stiffness matrix at the beginning of the analysis step serving as the Jacobian in the scheme. We expand the residual vector in the modified Newton method using a Fourier series of the same form as the displacement solution:

$$\overline{R}\,(t) \;= R_0 + \sum_{k=1}^{n} \left[R_k^s \sin k\omega t + R_k^c \cos k\omega t \right],$$

where each residual vector coefficient R_0, R_k^s, and R_k^c in the Fourier series corresponds to a displacement coefficient u_0, u_k^s, and u_k^c, respectively. The residual coefficients are obtained by tracking through the entire load cycle. At each instant in time in the cycle Abaqus/Standard obtains the residual vector $R(t)$ by using standard element-by-element calculations, which—when integrated over the entire cycle—provide the Fourier coefficients

$$\begin{aligned}
R_0 &= \frac{1}{T}\int_0^T R(t)\; dt, \\
R_k^s &= \frac{2}{T}\int_0^T R(t) \sin k\omega t\; dt, \\
R_k^c &= \frac{2}{T}\int_0^T R(t) \cos k\omega t\; dt.
\end{aligned}$$

The displacement solution is obtained by solving for corrections to the displacement Fourier coefficients corresponding to each residual coefficient. The updated displacement solution is used in the next iteration to obtain the displacements at each instant in time. This process is repeated until convergence is obtained. Each pass through the complete load cycle can, therefore, be thought of as a single iteration of the solution to the nonlinear problem. Convergence is measured by ensuring that all entries of the residual coefficients are small.

The algorithm to obtain a stabilized cycle is described in detail in *Direct cyclic algorithm*.

Direct Cyclic Analysis

A direct cyclic step can be the only step in an analysis, can follow a general or linear perturbation step, or can be followed by a general or linear perturbation step. If a direct cyclic step is followed by a general step, the solution at the end of the direct cyclic step will be the initial state of the general step. If a direct cyclic step follows a general or linear perturbation step, the elastic stiffness matrix at the end of the last general analysis step prior to the direct cyclic step will serve as the Jacobian in the direct cyclic procedure. Any prior (non-cyclic) loads are simply included in the constant part of the Fourier expansion of the residual vectors, and the plastic strains at the end of the preloading step are used as initial conditions for the direct cyclic step.

Multiple direct cyclic analysis steps can be included in a single analysis. In such a case the Fourier series coefficients obtained in the previous step can be used as starting values in the current step. By default, the Fourier coefficients are reset to zero, thus allowing application of cyclic loading conditions that are very different from those defined in the previous direct cyclic step.

You can specify that a direct cyclic step in a restart analysis should use the Fourier coefficients from the previous step, thus allowing continuation of an analysis that has not reached a stabilized cycle. In a direct cyclic analysis a restart file is written at the end of the cycle or time period. Consequently, a restart analysis that is a continuation of a previous direct cyclic analysis will start with a new iteration at $t = 0$ (see *Restarting an Analysis*).

Input File Usage: Use the following option to reset the Fourier series coefficients to zero:

**DIRECT CYCLIC*, CONTINUE=NO (default)

Use the following option to specify that the current step is a continuation of the previous direct cyclic step:

**DIRECT CYCLIC*, CONTINUE=YES

Abaqus/CAE Usage: Use the following option to reset the Fourier series coefficients to zero (default):

Step module: **Create Step**: **General**: **Direct cyclic**

Use the following option to specify that the current step is a continuation of the previous direct cyclic step:

Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Use displacement Fourier coefficients from previous direct cyclic step**

Using the Direct Cyclic Approach to Perform Low-Cycle Fatigue Analysis

The direct cyclic procedure can also be used in conjunction with the damage extrapolation technique to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding at the interfaces in laminated composites in a low-cycle fatigue analysis. In this case multiple cycles can be included in a single direct cyclic analysis, as described in *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*.

Input File Usage: **DIRECT CYCLIC*, FATIGUE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**

Controlling the Solution Accuracy

Direct cyclic analysis combines a Fourier series approximation with time integration of the nonlinear material behavior to obtain the stabilized cyclic solution iteratively using a modified Newton method. The accuracy of the algorithm depends on the number of Fourier terms used, the number of iterations taken to obtain the stabilized solution, and the number of time points within the load period at which the material response and residual vector are evaluated. Abaqus/Standard allows you to control the solution in several ways, as described below.

Controlling the Iterations in the Modified Newton Method

In the direct cyclic method global Newton iterations are performed to determine corrections to the displacement Fourier coefficients. During each global iteration Abaqus/Standard tracks through the entire time cycle to compute the residual vector at a suitable number of time points. This involves standard element-by-element finite element calculations in which history-dependent material variables are integrated. The residual vector is integrated over the period to obtain the Fourier residual coefficients, which in turn yield corrections in displacement coefficients when the system of equations is solved. Abaqus/Standard will continue with the iterative process until convergence is obtained or until the maximum number of iterations allowed has been reached. You can specify the maximum number of iterations when you define the direct cyclic step; the default is 200 iterations.

Input File Usage: **DIRECT CYCLIC*
, , , , , , , *max number of iterations*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Maximum number of iterations:** *max number of iterations*

Specifying Convergence Criteria

Convergence is best measured by ensuring that all the residual coefficients are sufficiently small compared to the time averaged force and that all the corrections to displacement Fourier coefficients are sufficiently small compared to the displacement Fourier coefficients. The time averaged force is defined in *Convergence Criteria for Nonlinear Problems*. Abaqus/Standard requires that the ratio of the maximum residual coefficient to the time averaged force, CR_n^α, and the ratio of the maximum correction to the displacement coefficients to the largest displacement coefficient, CU_n^α, are less than the tolerances. The default values are CR_n^α = 0.005 and CU_n^α = 0.005. To change these values, you must define direct cyclic controls.

When a stabilized cyclic response does not exist, the method will not converge. In the case where plastic ratcheting occurs, the displacement and residual coefficients of all the periodic terms (u_k^s, u_k^c, and R_k^c, R_k^c) in the Fourier series converge. However, the displacement and the residual coefficients of the constant term (u_0 and R_0) in the Fourier series continue to grow from one iteration to another iteration. The user-specified tolerances CR_0^α and CU_0^α are used to detect the plastic ratcheting. The default values are CR_0^α = 0.005 and CU_0^α = 0.005. For more information, see *Controlling the Solution Accuracy in Direct Cyclic Analysis*.

Input File Usage: **CONTROLS*, TYPE=DIRECT CYCLIC

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**; **Specify: Direct Cyclic**

Controlling the Fourier Representations

The number of Fourier terms required to obtain an accurate solution depends on the variation of the load as well as the variation of the structural response over the period. In determining the number of terms, keep in mind that the objective of this kind of analysis is to make low-cycle fatigue predictions. Hence, the goal is to obtain good approximation of the plastic strain cycle at each point; local inaccuracies in the stresses are less important. More Fourier terms usually provide a more accurate solution but at the expense of additional data storage and computational time. In addition, an accurate integration of the Fourier residual coefficients requires that the residual vector be evaluated at an adequate number of time points during the cycle. Abaqus/Standard uses a trapezoidal rule, which assumes a linear variation of the residual over a time increment, to integrate the residual coefficients. For accurate integration the number of time points must be larger than the number of Fourier coefficients (which is equal to $2n+1$, where n represents the number of Fourier terms). Abaqus/Standard will automatically reduce the number of Fourier coefficients used for the next iteration if it is found to be greater than the number of increments taken to complete an iteration.

Abaqus/Standard uses an adaptive algorithm to determine the number of Fourier terms. By default, Abaqus/Standard starts with 11 terms and determines the response of the structure by using the iterative method described before. Once convergence is obtained (which is measured by ensuring that all the residual vector coefficients and all the corrections to displacement coefficients in the Fourier series are sufficiently small), Abaqus/Standard evaluates if a sufficient number of Fourier terms are used by determining if equilibrium was satisfied at all the time points during the cycle. If equilibrium is satisfied at all time points, the solution is accepted. Otherwise, Abaqus/Standard increases the number of Fourier terms (by default, 5 terms are added) and continues with the iterative scheme until convergence with the new number of Fourier terms is obtained. This process is repeated until equilibrium is reached or until the maximum number of Fourier terms has been used. This scheme is best illustrated in *Figure 2*, where both local equilibrium and overall convergence are obtained when the number of Fourier terms is equal to 21. A maximum number of 25 Fourier terms is used by default. You can specify the initial and maximum number of Fourier terms and the increment in the number of terms when you define the direct cyclic step.

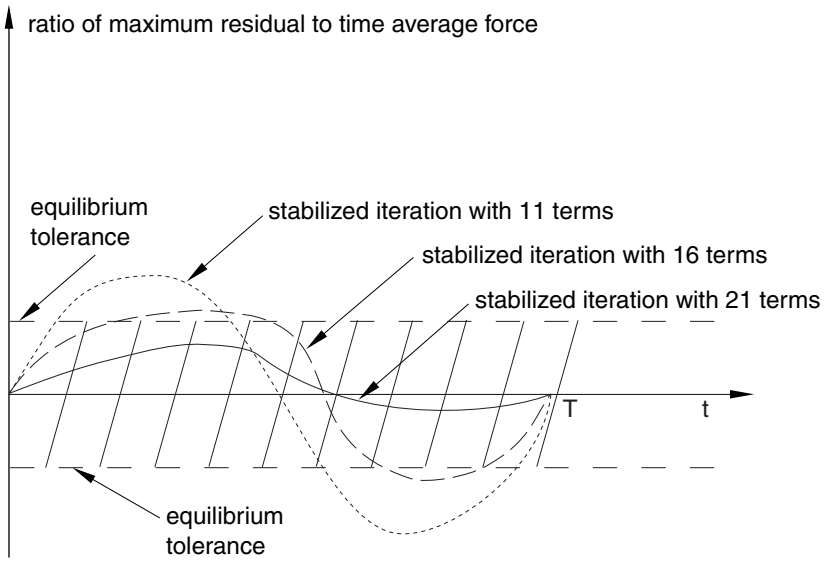


Figure 2: Stabilized iterations with different Fourier terms.

You can also define the convergence criteria for determining convergence and for determining whether equilibrium is achieved at all time points through the period (see *Commonly Used Control Parameters*), with suitable defaults set by Abaqus/Standard.

In a direct cyclic analysis that has not reached a stabilized cycle, you can increase the number of iterations or Fourier terms upon restart, thus allowing continuation of an analysis.

Abaqus/Standard provides detailed output of the maximum residual at each time point, the maximum residual coefficient, the maximum displacement coefficient, the maximum correction to displacement coefficients, and the number of Fourier terms at the end of each iteration in the message (`.msg`) file. This output is described in more detail below.

Input File Usage: **DIRECT CYCLIC*
, , , , *initial number of terms*, *max number of terms*, *increment in number of terms*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Number of Fourier terms: Initial:** *initial number of terms*, **Maximum:** *max number of terms*, **Increment:** *increment in number of terms*

Controlling the Incrementation during the Cyclic Time Period

To ensure an accurate solution, the material history as well as the residual vector must be evaluated at a sufficient number of time points during the cycle. The number of time points, n_T, at which the response is computed must be larger than the number of Fourier coefficients; i.e., $n_T > 2n + 1$. Abaqus/Standard will automatically adjust the number of Fourier coefficients if such a condition is not satisfied. You can specify the time incrementation over the cycle directly, or it can be determined automatically by Abaqus/Standard.

You should specify the maximum number of increments allowed in the time period as part of the step definition. The default is 100.

Automatic Incrementation

There are several ways to choose the automatic incrementation scheme. If you specify only the maximum allowable nodal temperature change in an increment, the time increments are selected automatically based on this value. Abaqus/Standard will restrict the time increments to ensure that the maximum temperature change is not exceeded at any node during any increment of the analysis.

For rate-dependent constitutive equations you can limit the size of the time increment by the accuracy of the integration. The user-specified accuracy tolerance parameter limits the maximum inelastic strain rate change allowed over an increment:

$$tolerance \geq \left(\dot{\bar{\varepsilon}}^{cr} \big|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr} \big|_{t} \right) \Delta t,$$

where t is the time at the beginning of the increment, Δt is the time increment (so that $t + \Delta t$ is the time at the end of the increment), and $\dot{\bar{\varepsilon}}^{cr}$ is the equivalent creep strain rate. To achieve sufficient accuracy, the value chosen for the accuracy tolerance parameter should be on the order of σ_{err}/E for creep problems, where σ_{err} is an acceptable level of error in the stress and E is a typical elastic modulus, or on the order of the elastic strains for viscoelasticity problems.

If rate-dependent constitutive equations are used in combination with a varying temperature, both controls can be used simultaneously. Abaqus/Standard will then choose the increments that satisfy both criteria.

If the time integration accuracy measure specified by either or both of the above controls is satisfied after I_T consecutive increments without cutbacks, the next time increment will be increased by a factor of D_M. Both I_T and D_M are user-defined parameters (see *Increasing the Time Increment Size*). The defaults are I_T = 3 and D_M = 1.5.

Input File Usage: Use the following option to specify the maximum allowable nodal temperature change:

**DIRECT CYCLIC*, DELTMX=$\Delta\theta_{max}$

Use the following option to specify the accuracy tolerance parameter:

**DIRECT CYCLIC*, CETOL=*tolerance*

Abaqus/CAE Usage: Use the following option to specify the maximum allowable nodal temperature change:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Use the following option to specify the accuracy tolerance parameter:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Fixed Time Incrementation

If neither the accuracy tolerance parameter nor the maximum allowable nodal temperature change is specified, the size of the time increment is fixed. You must specify the time increment Δt and the time period T.

Input File Usage: **DIRECT CYCLIC*
Δt, T

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Cycle time period:** T; **Incrementation: Type: Fixed**, **Increment size:** Δt

Defining the Time Points at Which the Response Must Be Evaluated

The user-defined time incrementation for a direct cyclic step can be augmented or superseded by specifying particular time points in the loading history at which the response of the structure should be evaluated. This feature is particularly useful if you know prior to the analysis at which time points in the analysis the load reaches a maximum and/or minimum value or when the response will change rapidly. An example is the analysis of the heating/cooling thermal cycle of an engine component where you typically know when the temperature reaches a maximum value.

When time points are used with fixed time incrementation, the time incrementation specified for the direct cyclic step is ignored and instead the time incrementation precisely follows the specified time points. If time points are used with automatic incrementation, the time incrementation is variable; but the response of the structure will be evaluated at the specified time points.

The time points can be listed individually, or they can be generated automatically by specifying the starting time point, ending time point, and increment in time between the two specified time points.

Input File Usage: Use the following options to list time points individually:

**TIME POINTS*, NAME=*time points name*
**DIRECT CYCLIC*, TIME POINTS=*time points name*

Use the following options to generate time points automatically:

**TIME POINTS*, NAME=*time points name*, GENERATE
**DIRECT CYCLIC*, TIME POINTS=*time points name*

Abaqus/CAE Usage: Use the following options to list time points individually:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Evaluate structure response at time points:** *time points name*

Use the following options to generate time points automatically:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Evaluate structure response at time points: Create**; **Edit Time Points: Specify using delimiters: Start**, **End**, **Increment**

Controlling the Application of Periodicity Conditions

By default, Abaqus/Standard imposes periodic conditions during the iterative solution process by using the state obtained at the end of the previous iteration as the starting state for the current iteration; i.e., $s_{t=0}^{i+1} = s_{t=T}^{i}$, where s is a solution variable such as plastic strain.

In cases where the periodic solution is not easily found (for example, when the loading is close to causing ratcheting), the state around which the periodic solution is obtained may show considerably more "drift" than would be obtained in a transient analysis. In such cases you may wish to delay the application of periodic conditions as an artificial method to reduce this drift. *Figure 3* compares the response of two identical structures subjected to the same set of cyclic loads and boundary conditions, where each structure experienced a different loading history prior to the application of the cyclic loads. *Figure 3* shows that the prior loading history only affects the mean value of stress and strain; it does not affect the shape of the stress-strain curves or the amount of energy dissipated during the cycle.

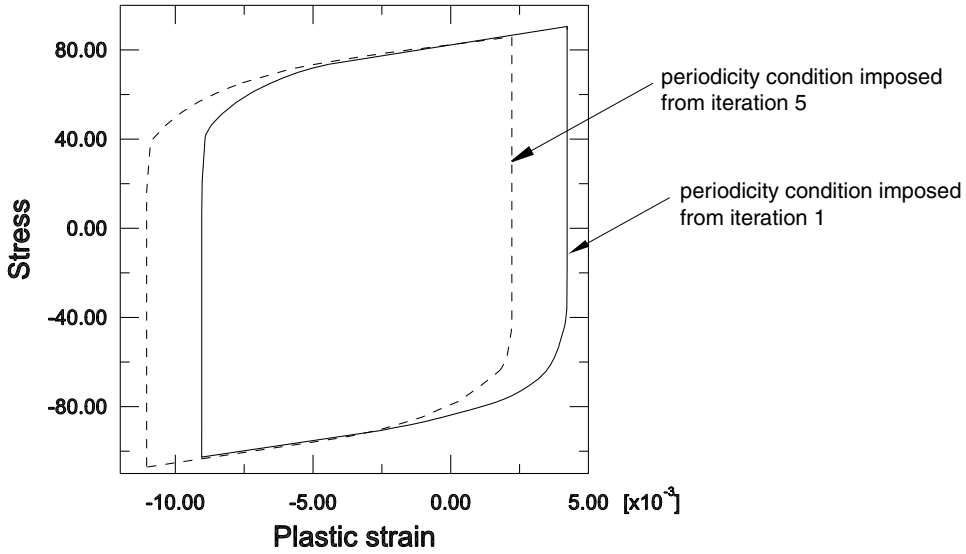


Figure 3: Influence of periodicity condition on mean value of the strains over a stabilized cycle.

By delaying the application of periodicity conditions, you can influence the mean stress and strain level. However, this is rarely necessary since the average stress and strain levels are usually not needed for low-cycle fatigue life predictions.

You can control when the periodicity conditions are applied by defining direct cyclic controls to specify the variable I_{PI}. This variable defines from which iteration onward the application of periodic conditions will be activated. For example, setting $I_{PI} = 6$ means that the periodicity conditions are applied from iteration 6 onwards. The default is $I_{PI} = 1$, which is appropriate for most analyses.

Input File Usage: **CONTROLS*, TYPE=DIRECT CYCLIC
I_{PI}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**; **Direct Cyclic:** I_{PI}

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see *Initial Conditions*).

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom. During the analysis, prescribed boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the analysis consists of several steps, the usual rules apply (see *Boundary Conditions*). At each new step the boundary condition can either be modified or completely defined. All boundary conditions defined in previous steps remain unchanged unless they are redefined.

Loads

The following loads can be prescribed in a direct cyclic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

During the analysis each load must have an amplitude definition that is cyclic over the step where the start value must be equal to the end value (see *Amplitude Curves*). If the analysis consists of several steps, the usual rules apply (see *About Loads*). At each new step the loading can either be modified or completely defined. All loads defined in previous steps remain unchanged unless they are redefined.

Predefined Fields

The following predefined fields can be specified in a direct cyclic analysis, as described in *Predefined Fields*:

- Temperature is not a degree of freedom in a direct cyclic analysis, but nodal temperatures can be specified as a predefined field. The temperature values specified must be cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see *Initial Conditions*). Alternatively, you can ramp the temperatures back to their initial condition values, as described in *Predefined Fields*. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. The field variable values specified must be cyclic over the step.

Material Options

Most material models, including user-defined materials (defined using user subroutine *UMAT*), that describe mechanical behavior are available for use in a direct cyclic analysis.

The following material properties are not active during a direct cyclic analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, piezoelectric properties, and pore fluid flow properties.

Rate-dependent yield (*Rate-Dependent Yield*), rate-dependent creep (*Rate-Dependent Plasticity: Creep and Swelling*), and two-layer viscoplasticity (*Two-Layer Viscoplasticity*) can also be used during a direct cyclic analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a direct cyclic analysis (see *Choosing the Appropriate Element for an Analysis Type*).

Output

Different types of output are available for postprocessing and for monitoring a direct cyclic analysis.

Message File Information

Abaqus/Standard prints the residual force, time average force, and a flag to indicate if equilibrium was satisfied in the message (`.msg`) file at different time increments for each iteration. You can control the frequency in increments at which information is printed to the message file, and you can suppress the output; the default is to print output every 10 increments (see *The Abaqus/Standard Message File* for more information).

Abaqus/Standard also prints the number of Fourier terms used, the maximum residual coefficient, the maximum correction to displacement coefficients, and the maximum displacement coefficient in the Fourier series in the message file at the end of each iteration. An example of the output is shown below:

```
                     ITERATION    26 STARTS
INC     TIME         STEP        LARG. RESI.    TIME AVG.    FORCE
        INC          TIME        FORCE          FORCE        EQUV.
10      0.250        2.50        1.008E+01      50.9          N
20      0.250        5.00        1.622E+01      76.8          N
30      0.250        7.50        4.622E-02      99.8          Y

                     ITERATION    26 SUMMARY
NUMBER OF FOURIER TERMS USED 40, TOTAL NUMBER OF INCREMENTS  120
CYCLE/STEP TIME   30.0,    TOTAL TIME COMPLETED       31.0
AVERAGE FORCE     21.2     TIME AVG. FORCE     25.7

MAX. COEFFICIENT OF DISP.                    0.142  AT NODE 24 DOF 2
MAX. COEFF. OF RESI. FORCE ON CONST. TERM    31.7  AT NODE 44 DOF 1
MAX. COEFF. OF RESI. FORCE ON PERI. TERMS    0.82  AT NODE  6 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON CONST. TERM 0.002 AT NODE 50 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON PERI. TERMS 0.015 AT NODE 50 DOF 3
```

Results Output

Element and nodal output are written only when the stabilized cycle is reached. If a stabilized cycle has not been reached at the end of an analysis, output is written for the last iteration of the step. The element output available for a direct cyclic analysis includes stress; strain; energies; and the values of state, field, and user-defined variables. All the energies are set equal to zero at the beginning of each iteration since energies dissipated over an entire stabilized cycle are of interest in making fatigue life predictions in direct cyclic analysis. The nodal output available includes displacements, reaction forces, and coordinates. All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Recovering Additional Results for an Iteration

You may want to recover additional results for an iteration rather than for the stabilized cycle. You can extract these results from the restart data (see *Recovering Additional Results Output from Restart Data in Abaqus/Standard*). This feature is particularly useful if you want to evaluate the shift of the strain from one iteration to another iteration when plastic ratcheting occurs.

Input File Usage: **POST OUTPUT*, ITERATION=*n*

Abaqus/CAE Usage: Recovering additional results for an iteration is not supported in Abaqus/CAE.

Specifying Output at Exact Times

Output at exact times is not supported for direct cyclic analysis. If output at exact times is requested, Abaqus will issue a warning message and change the output to an output at approximate times.

Limitations

A direct cyclic analysis is subject to the following limitations:

- Contact conditions cannot change during a direct cyclic analysis; they remain as they were defined at the beginning of the analysis or at the end of any general step prior to the direct cyclic step. Frictional slipping is not allowed during direct cyclic analyses; all points in contact are assumed to be sticking if friction is present.
- A direct cyclic step is always performed using the original coordinates of a model, even when the direct cyclic step follows a geometrically nonlinear step. To perform a direct cyclic analysis on the updated coordinates, you can use the import capability to import both the current state as well as the current configuration from the end of the desired geometrically nonlinear step.

Input File Template

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,INC=)
Set INC equal to the maximum number of increments in a single loading cycle
*DIRECT CYCLIC
```

```
Data line to define time increment, cycle time, initial number of Fourier terms,
maximum number of Fourier terms, increment in number of Fourier terms,
and maximum number of iterations
*TIME POINTS
Data lines to list time points
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
*END STEP
**
*STEP(,INC=)
*DIRECT CYCLIC, DELTMX
Data line to control automatic time incrementation and Fourier representations
*BOUNDARY, OP=MOD,AMPLITUDE=
Data lines to modify or add zero-valued or nonzero boundary conditions
*CLOAD, OP=NEW, AMPLITUDE=
Data lines to specify new concentrated loads; all previous concentrated
loads will be removed
*DLOAD, OP=MOD, AMPLITUDE=
Data lines to specify additional or modified distributed loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify additional or modified values of predefined fields
*END STEP
```

Eigenvalue Buckling Prediction

Products: Abaqus/Standard Abaqus/CAE

Overview

Eigenvalue buckling analysis:

- is generally used to estimate the critical (bifurcation) load of “stiff” structures;
- is a linear perturbation procedure;
- can be the first step in an analysis of an unloaded structure, or it can be performed after the structure has been preloaded—if the structure has been preloaded, the buckling load from the preloaded state is calculated;
- can be used in the investigation of the imperfection sensitivity of a structure;
- works only with symmetric matrices (hence, unsymmetric stiffness contributions such as the load stiffness associated with follower loads are symmetrized); and
- cannot be used in a model containing substructures.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Static Stress Analysis Procedures*
- **BUCKLE*
- *Configuring a buckling procedure*
- *Creating and modifying prescribed conditions*

General Eigenvalue Buckling

In an eigenvalue buckling problem we look for the loads for which the model stiffness matrix becomes singular, so that the problem

$$K^{MN}v^{M} = 0$$

has nontrivial solutions. K^{MN} is the tangent stiffness matrix when the loads are applied, and the v^{M} are nontrivial displacement solutions. The applied loads can consist of pressures, concentrated forces, nonzero prescribed displacements, and/or thermal loading.

Eigenvalue buckling is generally used to estimate the critical buckling loads of stiff structures (classical eigenvalue buckling). Stiff structures carry their design loads primarily by axial or membrane action, rather than by bending action. Their response usually involves very little deformation prior to buckling. A simple example of a stiff structure is the Euler column, which responds very stiffly to a compressive axial load until a critical load is reached, when it bends suddenly and exhibits a much lower stiffness. However, even when the response of a structure is nonlinear before collapse, a general eigenvalue buckling analysis can provide useful estimates of collapse mode shapes.

The Base State

The buckling loads are calculated relative to the base state of the structure. If the eigenvalue buckling procedure is the first step in an analysis, the initial conditions form the base state; otherwise, the base state is the current state of the model at the end of the last general analysis step (see *General and Perturbation Procedures*).

Therefore, the base state can include preloads ("dead" loads), $\boldsymbol{P^N}$. The preloads are often zero in classical eigenvalue buckling problems.

If geometric nonlinearity was included in the general analysis steps prior to the eigenvalue buckling analysis (see *General and Perturbation Procedures*), the base state geometry is the deformed geometry at the end of the last general analysis step. If geometric nonlinearity was omitted, the base state geometry is the original configuration of the body.

The Eigenvalue Problem

An incremental loading pattern, $\boldsymbol{Q^N}$, is defined in the eigenvalue buckling prediction step. The magnitude of this loading is not important; it will be scaled by the load multipliers, $\boldsymbol{\lambda_i}$, found in the eigenvalue problem:

$$\left(K_0^{NM} + \lambda_i K_\Delta^{NM}\right) v_i^M = 0,$$

where

$\boldsymbol{K_0^{NM}}$

is the stiffness matrix corresponding to the base state, which includes the effects of the preloads, $\boldsymbol{P^N}$ (if any);

$\boldsymbol{K_\Delta^{NM}}$

is the differential initial stress and load stiffness matrix due to the incremental loading pattern, $\boldsymbol{Q^N}$;

$\boldsymbol{\lambda_i}$

are the eigenvalues;

$\boldsymbol{v_i^M}$

are the buckling mode shapes (eigenvectors);

M* and *N

refer to degrees of freedom *M* and *N* of the whole model; and

$\boldsymbol{i}$

refers to the *i*th buckling mode.

The critical buckling loads are then $P^N + \lambda_i Q^N$. Normally, the lowest value of λ_i is of interest. The preload pattern, P^N, and perturbation load pattern, Q^N, may be different. For example, P^N might be thermal loading caused by temperature changes, while Q^N is caused by application of pressure.

The buckling mode shapes, v_i^M, are normalized vectors and do not represent actual magnitudes of deformation at critical load. They are normalized so that the maximum displacement component is 1.0. If all displacement components are zero, the maximum rotation component is normalized to 1.0. These buckling mode shapes are often the most useful outcome of the eigenvalue analysis, since they predict the likely failure mode of the structure.

Abaqus/Standard can extract eigenvalues and eigenvectors for symmetric matrices only; therefore, K_0^{NM} and K_Δ^{NM} are symmetrized. If the matrices have significant unsymmetric parts, the eigenproblem may not be exactly what you expected to solve.

Selecting the Eigenvalue Extraction Method

Abaqus/Standard offers the Lanczos and the subspace iteration eigenvalue extraction methods. The Lanczos method is generally faster when a large number of eigenmodes is required for a system with many degrees of freedom. The subspace iteration method may be faster when only a few (less than 20) eigenmodes are needed.

By default, the subspace iteration eigensolver is employed. You can use subspace iteration and the Lanczos solver for different steps in the same analysis; there is no requirement that you use the same eigensolver for all appropriate steps.

For both eigensolvers you specify the desired number of eigenvalues; Abaqus/Standard chooses a suitable number of vectors for the subspace iteration procedure or a suitable block size for the Lanczos method (although you can override this choice, if needed). Significant overestimation of the actual number of eigenvalues can create very large files. If the actual number of eigenvalues is underestimated, Abaqus/Standard issues a corresponding warning message.

In general, the block size for the Lanczos method should be as large as the largest expected multiplicity of eigenvalues (that is, the largest number of modes with the same eigenvalue). A block size larger than 10 is not recommended. If the number of eigenvalues requested is *n*, the default block size is the minimum of (7, *n*). The number of block Lanczos steps is usually determined by Abaqus/Standard, but you can change it when you define the eigenvalue buckling prediction step. In general, if a particular type of eigenproblem converges slowly, providing more block Lanczos steps will reduce the analysis cost. However, if you know that a particular type of problem converges quickly, providing fewer block Lanczos steps will reduce the amount of in-core memory used. If the number of eigenvalues requested is *n*, the default is

Block size	$n \leq 10$	$n > 10$
1	40	70
2	40	60
3	30	60
4	30	30

If you use the subspace iteration technique, you can also specify the maximum eigenvalue of interest; Abaqus/Standard extracts eigenvalues until either the requested number of eigenvalues has been extracted or the last eigenvalue extracted exceeds the maximum eigenvalue of interest.

If you use the Lanczos eigensolver, you can also specify the minimum and/or maximum eigenvalues of interest; Abaqus/Standard extracts eigenvalues until either the requested number of eigenvalues has been extracted in the given range or all the eigenvalues in the given range have been extracted.

Input File Usage: Use the following option to perform an eigenvalue buckling analysis using the subspace iteration method:

**BUCKLE*, EIGENSOLVER=SUBSPACE (default)

Use the following option to perform an eigenvalue buckling analysis using the Lanczos method:

**BUCKLE*, EIGENSOLVER=LANCZOS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Buckle**: **Eigensolver: Lanczos** or **Subspace**

Limitations Associated with Applying the Lanczos Eigensolver to a Buckling Analysis

The Lanczos eigensolver cannot be used for buckling analyses in which the stiffness matrix is indefinite, as in the following cases:

- A model containing hybrid elements or connector elements.
- A model containing distributing coupling constraints, defined either directly (*Coupling Constraints*, *Shell-to-Solid Coupling*, or *Mesh-Independent Fasteners*) or by the distributing coupling elements (DCOUP2D and DCOUP3D).
- A model containing contact pairs or contact elements.
- A model that has been preloaded above the bifurcation (buckling) load.
- A model that has rigid body modes.

In such cases Abaqus/Standard will issue an error message and terminate the analysis.

Order of Calculation and Formation of the Stiffness Matrices

In an eigenvalue buckling prediction step Abaqus/Standard first does a static perturbation analysis to determine the incremental stresses, $\Delta\boldsymbol{\sigma}$, due to $\boldsymbol{Q}^N$. If the base state did not include geometric nonlinearity, the stiffness matrix used in this static perturbation analysis is the tangent elastic stiffness. If the base state did include geometric nonlinearity, initial stress and load stiffness terms (due to the preload, $\boldsymbol{P}^N$) are included. The stiffness matrix $\boldsymbol{K}_{\Delta}^{NM}$ corresponding to $\Delta\boldsymbol{\sigma}$ and $\boldsymbol{Q}^N$ is then formed.

In the eigenvalue extraction portion of the buckling step, the stiffness matrix $\boldsymbol{K}_{0}^{NM}$ corresponding to the base state geometry is formed. Initial stress and the load stiffness terms due to the preload, $\boldsymbol{P}^N$, are always included regardless of whether or not geometric nonlinearity is included and are calculated based on the geometry of the base state.

When forming the stiffness matrices $\boldsymbol{K}_{0}^{NM}$ and $\boldsymbol{K}_{\Delta}^{NM}$, all contact conditions are fixed in the base state.

Buckling Modes with Closely Spaced Eigenvalues

Some structures have many buckling modes with closely spaced eigenvalues, which can cause numerical problems. In these cases it often helps to apply enough preload, $\boldsymbol{P}^N$, to load the structure to just below the buckling load before performing the eigenvalue extraction.

If $P^N = \mu Q^N$—where μ is a scalar constant and the structure is "stiff" and elastic—and if the problem is linear, the structural stiffness changes to $K_0^{NM} + \mu K_\Delta^{NM}$ and the buckling loads are given by $(\mu + \lambda_i) Q^N$. The process is equivalent to a dynamic eigenfrequency extraction with shift μ. The structure should not be preloaded above the buckling load. In that case the subspace iteration process may fail to converge or produce incorrect results; the Lanczos eigensolver cannot be used (as discussed earlier).

In many cases a series of closely spaced eigenvalues indicates that the structure is imperfection sensitive. An eigenvalue buckling analysis will not give accurate predictions of the buckling load for imperfection-sensitive structures; the static Riks procedure should be used instead (see *Unstable Collapse and Postbuckling Analysis*).

Understanding Negative Eigenvalues

Sometimes, negative eigenvalues are reported in an eigenvalue buckling analysis. In most cases such negative eigenvalues indicate that the structure would buckle if the load were applied in the opposite direction. A classical example is a plate under shear loading; the plate will buckle at the same value for positive and negative applied shear load. Buckling under reverse loading can also occur in situations where it may not be expected. For example, a pressure vessel under external pressure may exhibit a negative eigenvalue (buckling under internal pressure) due to local buckling of a stiffener. Such "physical" negative buckling modes are usually readily understood once they are displayed and can usually be avoided by applying a preload before the buckling analysis.

Negative eigenvalues sometimes correspond to buckling modes that cannot be understood readily in terms of physical behavior, particularly if a preload is applied that causes significant geometric nonlinearity. In this case a geometrically nonlinear load-displacement analysis should be performed (*Unstable Collapse and Postbuckling Analysis*).

Including Large Geometry Changes in a Buckling Analysis

Because buckling analysis is usually done for "stiff" structures, it is not usually necessary to include the effects of geometry change in establishing equilibrium for the base state. However, if significant geometry change is involved in the base state and this effect is considered to be important, it can be included by specifying that geometric nonlinearity should be considered for the base state step (see *General and Perturbation Procedures*). In such cases it is probably more realistic to perform a geometrically nonlinear load-displacement analysis (Riks analysis) to determine the collapse loads, especially for imperfection-sensitive structures.

While large deformation can be included in the preload, the eigenvalue buckling theory relies on there being little geometric change due to the "live" buckling load, $\lambda_i Q^N$. If the live load produces significant geometric change, a nonlinear collapse (Riks) analysis must be used. The total buckling load predicted by the eigenvalue analysis, $P^N + \lambda_i Q^N$, may be a good estimate for the limit load in the nonlinear buckling analysis. The Riks method is described in *Unstable Collapse and Postbuckling Analysis*.

Initial Conditions

The initial values of quantities such as stress, temperature, field variables, and solution-dependent variables can be specified for an eigenvalue buckling analysis. If the buckling step is the first step in the analysis, these initial conditions form the base state of the structure. *Initial Conditions* describes all of the available initial conditions.

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) or to warping degree of freedom 7 in open-section beam elements (*Boundary Conditions*). A nonzero prescribed boundary condition in a general analysis step preceding the eigenvalue buckling analysis can be used to preload the structure. Nonzero boundary conditions prescribed in an eigenvalue buckling step will contribute to the incremental stress

$\Delta\boldsymbol{\sigma}$ and, thus, will contribute to the differential initial stress stiffness. When prescribing nonzero boundary conditions, you must interpret the resulting eigenproblem carefully. Nonzero prescribed boundary conditions will be treated as constraints (i.e., as if they were fixed) during the eigenvalue extraction. Therefore, unless the prescribed boundary conditions are removed for the eigenvalue extraction by specifying buckling mode boundary conditions (see the discussion below), the mode shapes may be altered by these boundary conditions.

Amplitude definitions (*Amplitude Curves*) cannot be used to vary the magnitudes of prescribed boundary conditions during an eigenvalue buckling analysis.

You can define perturbation load and buckling mode boundary conditions in an eigenvalue buckling prediction step.

Input File Usage: Use either of the following two options to define perturbation load boundary conditions:

**BOUNDARY*
**BOUNDARY*, LOAD CASE=1

Use the following option to define buckling mode boundary conditions:

**BOUNDARY*, LOAD CASE=2, OP=NEW

The OP=NEW parameter is required when you define buckling mode boundary conditions in an eigenvalue buckling prediction step; however, the perturbation load boundary conditions in the step can use either OP=NEW or OP=MOD.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Mechanical** for the **Category** and **Symmetry/Antisymmetry/Encastre** for the **Types for Selected Step**: select region: toggle on **Stress perturbation only** to define a perturbation load boundary condition; toggle on **Buckling mode calculation only** to define a buckling mode boundary condition; toggle on **Stress perturbation and buckling mode calculation** to define both types of boundary conditions

Combining Boundary Conditions

The buckling mode shapes depend on the stresses in the base state as well as the incremental stresses due to the perturbation loading in the buckling step. These stresses are influenced by the boundary conditions used in each step. In a general eigenvalue buckling analysis the following types of boundary conditions can influence the stresses:

1. The boundary conditions in the base state.
2. The boundary conditions used to calculate the linear perturbation stresses, $\Delta\boldsymbol{\sigma}$. These boundary conditions will be:
   a. the perturbation load boundary conditions specified in the eigenvalue buckling step; or
   b. the base-state boundary conditions if no perturbation load boundary conditions are specified in the eigenvalue buckling step; or
   c. the buckling mode boundary conditions if neither perturbation load boundary conditions nor base-state boundary conditions exist.
3. The boundary conditions used for the eigenvalue extraction. These boundary conditions will be:
   a. the buckling mode boundary conditions; or

b. the perturbation load boundary conditions if buckling mode boundary conditions are not specified in the eigenvalue buckling step; or

c. the base-state boundary conditions if no boundary condition definition is used in the eigenvalue buckling step.

Table 1 summarizes the use of boundary conditions during an eigenvalue buckling step. When buckling mode boundary conditions are specified, *all* boundary conditions to be imposed during eigenvalue extraction must be specified.

Buckling of Symmetric Structures

The buckling mode shapes of symmetric structures subjected to symmetric loadings are either symmetric or antisymmetric. In such cases it is often more efficient to model only part of the structure and then perform the buckling analysis twice for each symmetry plane: once with symmetric boundary conditions and once with antisymmetric boundary conditions.

The live load pattern is usually symmetric, so symmetric boundary conditions are required for the calculation of the perturbation stresses used in the formation of the initial stress stiffness matrix. The boundary conditions must be switched to antisymmetric for the eigenvalue extraction to obtain the antisymmetric modes. *Buckling of a cylindrical shell under uniform axial pressure* illustrates such a case.

If the model includes more than one symmetry plane, it may be necessary to study all permutations of symmetric and antisymmetric boundary conditions for each symmetry plane.

Table 1: Boundary conditions in effect during the different portions of an eigenvalue buckling analysis.

User-defined boundary conditions		Boundary conditions used by Abaqus	
Base state	**Eigenvalue buckling prediction step**	**Linear perturbation**	**Eigenvalue extraction**
B	0	B	B
0	1	1	1
0	2	2	2
B	1	1	1
B	2	B	2
0	1, 2	1	2
B	1, 2	1	2
B = base-state boundary conditions; 0 = no boundary conditions specified			
1 = perturbation load boundary conditions			
2 = buckling mode boundary conditions			

Asymmetric Buckling of Axisymmetric Structures

Axisymmetric structures subjected to compressive loading often collapse in nonaxisymmetric modes. These modes cannot be found with purely axisymmetric modeling such as that provided by shell elements SAX1 and SAX2 (*Axisymmetric Shell Element Library*) or continuum elements CAX4 or CAX8 (*Axisymmetric Solid Element Library*). Such analyses must be done with three-dimensional shell or continuum elements.

Loads

The following types of loading can be prescribed in an eigenvalue buckling analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

The load stiffness can have a significant effect on the critical buckling load; therefore, Abaqus/Standard will take the load stiffness due to preloads into account when solving the eigenvalue buckling problem. It is important that the structure not be preloaded above the critical buckling load.

Any load applied during the eigenvalue buckling analysis is called a "live" load. This incremental load, $\boldsymbol{Q^N}$, describes the load pattern for which buckling sensitivity is being investigated; its magnitude is not important. This incremental loading definition represents linear perturbation loads, as described in *About Loads*.

Follower forces (such as concentrated loads assumed to rotate with the nodal rotation or pressure loads) lead to an unsymmetric load stiffness. Since eigenvalue extraction in Abaqus/Standard can be performed only on symmetric matrices, eigenvalue analysis with follower loads may not yield correct results.

Amplitude definitions cannot be used during an eigenvalue buckling analysis. *About Loads* describes all of the available loads.

Prescribed boundary conditions can also be used to load the structure in an eigenvalue buckling analysis, as discussed earlier.

Predefined Fields

In an eigenvalue buckling prediction step, nodal temperatures can be specified (see *Predefined Fields*). The specified temperatures can cause thermal strain during the static perturbation analysis if a thermal expansion coefficient is given for the material (see *Computing Thermal Strains in Linear Perturbation Steps*), and incremental stresses $\boldsymbol{\Delta\sigma}$ will be generated. Hence, Abaqus/Standard can analyze buckling due to thermal stress. The specified temperature will not affect temperature-dependent material properties during the eigenvalue buckling prediction step; the material properties are based on the temperature in the base state. Amplitude definitions cannot be used to vary the magnitudes of prescribed temperatures during an eigenvalue buckling analysis.

Material Options

During an eigenvalue buckling analysis, the model's response is defined by its linear elastic stiffness in the base state. All nonlinear and/or inelastic material properties, as well as effects involving time or strain rate, are ignored during an eigenvalue buckling analysis. In classical eigenvalue buckling the response in the base state is also linear.

If temperature-dependent elastic properties are used, the eigenvalue buckling analysis will not account for changes in the stiffness matrix due to temperature changes. The material properties of the base state will be used.

Acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties are not active during an eigenvalue buckling analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in an eigenvalue buckling analysis, with the exception that hybrid and contact

elements cannot be used with the Lanczos eigensolver (as discussed earlier). See *Choosing the Appropriate Element for an Analysis Type*.

Output

The values of the eigenvalues, λ_i, will be listed in the printed output file. If output of stresses, strains, reaction forces, etc. is requested, this information will be printed for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values. All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Buckling mode shapes can be plotted in the Visualization module of Abaqus/CAE.

Input File Template

The following template describes a very general eigenvalue buckling problem, where as many eigenvalue buckling prediction steps as needed can be specified.

Symmetric boundary conditions are specified in the model definition part of the Abaqus/Standard input and, therefore, belong to the base state (see *General and Perturbation Procedures*). In the first buckling step Abaqus/Standard uses the base-state boundary conditions to solve for the perturbation stresses as well as for the eigenvalue extraction.

In the second buckling step the boundary conditions for the base state, the initial stress calculation, and the eigenvalue extraction are all different. Abaqus/Standard uses the specified symmetry boundary conditions to solve for the perturbation stresses but uses the specified antisymmetry boundary conditions for the eigenvalue extraction.

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions contributing to the base state
**
*STEP, NLGEOM
The load stiffness terms will be included in the eigenvalue buckling steps
since the NLGEOM parameter is used in this (optional) preload step
*STATIC
Data line to control incrementation
*BOUNDARY
Data lines to specify nonzero boundary conditions (dead loads)
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify dead loads, P^N
*END STEP
**
*STEP
*BUCKLE
Data line to request the desired number of symmetric modes
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify perturbation loading, Q^N
*END STEP
**
*STEP
*BUCKLE
Data line to request the desired number of antisymmetric modes
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify perturbation loading, Q^N
*BOUNDARY, LOAD CASE=1
```

```
Data lines to specify all boundary conditions for perturbation loading
*BOUNDARY, LOAD CASE=2, OP=NEW
Data lines to specify all antisymmetric boundary conditions for eigenvalue extraction
*END STEP
```

Unstable Collapse and Postbuckling Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

The Riks method:

- is generally used to predict unstable, geometrically nonlinear collapse of a structure;
- can include nonlinear materials and boundary conditions;
- often follows an eigenvalue buckling analysis to provide complete information about a structure's collapse; and
- can be used to speed convergence of ill-conditioned or snap-through problems that do not exhibit instability.

References:

- *Defining an Analysis*
- *About Static Stress Analysis Procedures*
- *Introducing a Geometric Imperfection into a Model*
- **STATIC*
- **IMPERFECTION*
- *Configuring a static, Riks procedure*

Unstable Response

Geometrically nonlinear static problems sometimes involve buckling or collapse behavior, where the load-displacement response shows a negative stiffness and the structure must release strain energy to remain in equilibrium. Several approaches are possible for modeling such behavior. One is to treat the buckling response dynamically, thus actually modeling the response with inertia effects included as the structure snaps. This approach is easily accomplished by restarting the terminated static procedure (*Restarting an Analysis*) and switching to a dynamic procedure (*Implicit Dynamic Analysis Using Direct Integration*) when the static solution becomes unstable. In some simple cases displacement control can provide a solution, even when the conjugate load (the reaction force) is decreasing as the displacement increases. Another approach would be to use dashpots to stabilize the structure during a static analysis. Abaqus/Standard offers an automated version of this stabilization approach for the static analysis procedures (see *Static Stress Analysis*, *Quasi-Static Analysis*, *Fully Coupled Thermal-Stress Analysis*, or *Coupled Pore Fluid Diffusion and Stress Analysis*).

Alternatively, static equilibrium states during the unstable phase of the response can be found by using the "modified Riks method." This method is used for cases where the loading is proportional; that is, where the load magnitudes are governed by a single scalar parameter. The method can provide solutions even in cases of complex, unstable response such as that shown in *Figure 1*.

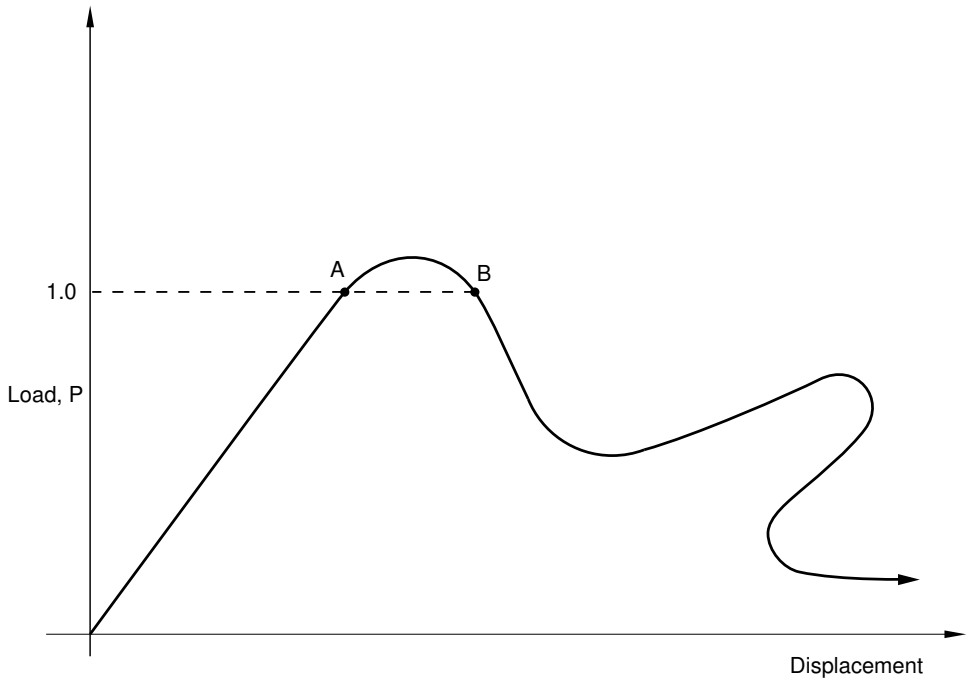


Figure 1: Proportional loading with unstable response.

The Riks method is also useful for solving ill-conditioned problems such as limit load problems or almost unstable problems that exhibit softening.

The Riks Method

In simple cases linear eigenvalue analysis (*Eigenvalue Buckling Prediction*) may be sufficient for design evaluation; but if there is concern about material nonlinearity, geometric nonlinearity prior to buckling, or unstable postbuckling response, a load-deflection (Riks) analysis must be performed to investigate the problem further.

The Riks method uses the load magnitude as an additional unknown; it solves simultaneously for loads and displacements. Therefore, another quantity must be used to measure the progress of the solution; Abaqus/Standard uses the "arc length," l, along the static equilibrium path in load-displacement space. This approach provides solutions regardless of whether the response is stable or unstable. See the *Modified Riks algorithm* for a detailed description of the method.

Proportional Loading

If the Riks step is a continuation of a previous history, any loads that exist at the beginning of the step and are not redefined are treated as "dead" loads with constant magnitude. A load whose magnitude is defined in the Riks step is referred to as a "reference" load. All prescribed loads are ramped from the initial (dead load) value to the reference values specified.

The loading during a Riks step is always proportional. The current load magnitude, $\boldsymbol{P}_{\mathbf{total}}$, is defined by

$$\boldsymbol{P}_{\mathbf{total}} = \boldsymbol{P}_0 + \boldsymbol{\lambda}\left(\boldsymbol{P}_{\mathbf{ref}} - \boldsymbol{P}_0\right),$$

where $\boldsymbol{P}_0$ is the "dead load," $\boldsymbol{P}_{\mathbf{ref}}$ is the reference load vector, and $\boldsymbol{\lambda}$ is the "load proportionality factor." The load proportionality factor is found as part of the solution. Abaqus/Standard prints out the current value of the load proportionality factor at each increment.

Incrementation

Abaqus/Standard uses Newton's method (as described in *Static Stress Analysis*) to solve the nonlinear equilibrium equations. The Riks procedure uses only a 1% extrapolation of the strain increment.

You provide an initial increment in arc length along the static equilibrium path, Δl_{in}, when you define the step. The initial load proportionality factor, $\Delta\lambda_{in}$, is computed as

$$\Delta\lambda_{in} = \frac{\Delta l_{in}}{l_{period}},$$

where l_{period} is a user-specified total arc length scale factor (typically set equal to 1). This value of $\Delta\lambda_{in}$ is used during the first iteration of a Riks step. For subsequent iterations and increments the value of λ is computed automatically, so you have no control over the load magnitude. The value of λ is part of the solution. Minimum and maximum arc length increments, Δl_{min} and Δl_{max}, can be used to control the automatic incrementation.

Input File Usage: **STATIC*, RIKS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**

Direct Incrementation

Direct user control of the increment size is also provided; in this case the incremental arc length, Δl, is kept constant. This method is not recommended for a Riks analysis since it prevents Abaqus/Standard from reducing the arc length when a severe nonlinearity is encountered.

Input File Usage: **STATIC*, RIKS, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**: **Incrementation**: **Type: Fixed**

Ending a Riks Analysis Step

Since the loading magnitude is part of the solution, you need a method to specify when the step is completed.

You can specify a maximum value of the load proportionality factor, λ_{end}, or a maximum displacement value at a specified degree of freedom. The step will terminate when either value is crossed. If neither of these finishing conditions is specified, the analysis will continue for the number of increments specified in the step definition (see *Defining an Analysis*).

Bifurcation

The Riks method works well in snap-through problems—those in which the equilibrium path in load-displacement space is smooth and does not branch. Generally you do not need take any special precautions in problems that do not exhibit branching (bifurcation). *Snap-through buckling analysis of circular arches* is an example of a smooth snap-through problem.

The Riks method can also be used to solve postbuckling problems, both with stable and unstable postbuckling behavior. However, the exact postbuckling problem cannot be analyzed directly due to the discontinuous response at the point of buckling. To analyze a postbuckling problem, it must be turned into a problem with continuous response instead of bifurcation. This effect can be accomplished by introducing an initial imperfection into a "perfect" geometry so that there is some response in the buckling mode before the critical load is reached.

Introducing Geometric Imperfections

Imperfections are usually introduced by perturbations in the geometry. Unless the precise shape of an imperfection is known, an imperfection consisting of multiple superimposed buckling modes must be introduced (*Eigenvalue Buckling Prediction*). Abaqus allows you to define imperfections; see *Introducing a Geometric Imperfection into a Model*.

In this way the Riks method can be used to perform postbuckling analyses of structures that show linear behavior prior to (bifurcation) buckling. An example of this method of introducing geometric imperfections is presented in *Buckling of a cylindrical shell under uniform axial pressure*.

By performing a load-displacement analysis, other important nonlinear effects, such as material inelasticity or contact, can be included. In contrast, all inelastic effects are ignored in a linear eigenvalue buckling analysis and all contact conditions are fixed in the base state. Imperfections based on linear buckling modes can also be useful for the analysis of structures that behave inelastically prior to reaching peak load.

Introducing Loading Imperfections

Perturbations in loads or boundary conditions can also be used to introduce initial imperfections. In this case fictitious “trigger” loads can be used to initiate the instability. The trigger loads should perturb the structure in the expected buckling modes. Typically, these loads are applied as dead loads prior to the Riks step so that they have fixed magnitudes. The magnitudes of trigger loads must be sufficiently small so that they do not affect the overall postbuckling solution. It is your responsibility to choose appropriate magnitudes and locations for such fictitious loads; Abaqus/Standard does not check that they are reasonable.

Obtaining a Solution at a Particular Load or Displacement Value

The Riks algorithm cannot obtain a solution at a given load or displacement value since these are treated as unknowns—termination occurs at the first solution that satisfies the step termination criterion. To obtain solutions at exact values of load or displacement, the solution must be restarted at the desired point in the step (*Restarting an Analysis*) and a new, non-Riks step must be defined. Since the subsequent step is a continuation of the Riks analysis, the load magnitude in that step must be given appropriately so that the step begins with the loading continuing to increase or decrease according to its behavior at the point of restart. For example, if the load was increasing at the restart point and was positive, a larger load magnitude than the current magnitude should be given in the restart step to continue this behavior. If the load was decreasing but positive, a smaller magnitude than the current magnitude should be specified.

Restrictions

A Riks analysis is subject to the following restrictions:

- A Riks step cannot be followed by another step in the same analysis. Subsequent steps must be analyzed by using the restart capability.
- If a Riks analysis includes irreversible deformation such as plasticity and a restart using another Riks step is attempted while the magnitude of the load on the structure is decreasing, Abaqus/Standard will find the elastic unloading solution. Therefore, restart should occur at a point in the analysis where the load magnitude is increasing if plasticity is present.
- For postbuckling problems involving loss of contact, the Riks method will usually not work; inertia or viscous damping forces (such as those provided by dashpots) must be introduced in a dynamic or static analysis to stabilize the solution.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified; *Initial Conditions* describes all of the available initial conditions.

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) or to warping degree of freedom 7 in open-section beam elements (*Boundary Conditions*). Amplitude definitions (*Amplitude Curves*) cannot be used to vary the magnitudes of prescribed boundary conditions during a Riks analysis.

Loads

The following loads can be prescribed in a Riks analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Since Abaqus/Standard scales loading magnitudes proportionally based on the user-specified magnitudes, amplitude references are ignored when the Riks method is chosen.

If follower loads are prescribed, their contribution to the stiffness matrix may be unsymmetric; the unsymmetric matrix storage and solution scheme can be used to improve computational efficiency in such cases (see *Defining an Analysis*).

Predefined Fields

Nodal temperatures can be specified (see *Predefined Fields*). Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The loads generated by the thermal strain contribute to the “reference” load specified for the Riks analysis and are ramped up with the load proportionality factor. Hence, the Riks procedure can analyze postbuckling and collapse due to thermal straining.

The values of other user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. Since the concept of time is replaced by arc length in a Riks analysis, the use of properties that change due to changes in temperatures and/or field variables is not recommended.

Material Options

Most material models that describe mechanical behavior are available for use in a Riks analysis. The following material properties are not active during a Riks analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties. Materials with history dependence can be used; however, it should be realized that the results will depend on the loading history, which is not known in advance.

The concept of time is replaced by arc length in a Riks analysis. Therefore, any effects involving time or strain rate (such as viscous damping or rate-dependent plasticity) are no longer treated correctly and should not be used.

See *Abaqus Materials Guide* for details on the material models available in Abaqus/Standard.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in a Riks analysis (see *Choosing the Appropriate Element for an Analysis Type*). Dashpots should not be used since velocities will be calculated as displacement increments divided by arc length, which is meaningless.

Output

Output options are provided to allow the magnitudes of individual load components (pressure, point loads, etc.) to be printed or to be written to the results file. The current value of the load proportionality factor, LPF, will be given automatically with any results or output database file output request. These output options are recommended when the Riks method is used so that load magnitudes can be seen directly. All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
…
*INITIAL CONDITIONS
Data lines to define initial conditions
*BOUNDARY
Data lines to specify zero-valued boundary conditions
**
*STEP, NLGEOM
*STATIC
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify preload (dead load), P_0
*END STEP
**
*STEP, NLGEOM
*STATIC, RIKS
Data line to define incrementation and stopping criteria
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify reference loading, P_ref
*END STEP
```

Linear Elastic Fatigue Crack Growth Analysis

Products: Abaqus/Standard

Overview

A fatigue crack growth analysis for linear elastic response:

- is a quasi-static analysis on a structure subjected to subcritical cyclic loading;
- is characterized by the fracture energy release rate;
- uses a classical incremental method for each loading cycle;
- does not make use of the Fourier representation of the displacement solutions as is the case for the direct cyclic framework;
- can be associated with thermal as well as mechanical loading;
- models propagation of a discrete crack along an arbitrary, solution-dependent path without remeshing in the bulk brittle material based on the principles of linear elastic fracture mechanics (LEFM) with the extended finite element method;
- models progressive delamination growth along a predefined path at the brittle material interfaces in laminated composites;
- models fatigue crack growths with the adaptive remeshing technique;
- uses the damage extrapolation technique to accelerate the fatigue crack growth analysis;
- accounts for the change of contact conditions and geometric nonlinearity; and
- can be simplified to accelerate the crack growth analysis in some special cases.

To simulate low-cycle fatigue of a ductile material, you should instead use progressive damage and failure modeling techniques (*Damage and Failure for Ductile Materials in Low-Cycle Fatigue Analysis*) with the direct cyclic procedure (*Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*).

References:

- *Defining an Analysis*
- *About Static Stress Analysis Procedures*
- *Crack Propagation Analysis*
- *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*
- *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*
- *Fatigue Crack Propagation with Adaptive Remeshing*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- *Damage and Failure for Ductile Materials in Low-Cycle Fatigue Analysis*
- **DEBOND*
- **FRACTURE CRITERION*
- **CONTROLS*

Define a Fatigue Crack Growth Analysis

The fatigue crack growth analysis capability in Abaqus/Standard is a quasi-static analysis on a structure subjected to subcritical cyclic loading. You can use the fatigue crack growth procedure to simulate two different classes of problems depending on the crack location.

At the brittle interface of laminated composites, the cyclic loading leads to interface strength degradation causing fatigue delamination growth. The onset and growth of delamination are characterized by the fracture energy release rate at the crack tip based on the Paris law (see *Paris, 1961*).

The other class of problems is for brittle bulk materials, in which the cyclic loading leads to material strength degradation causing fatigue crack growth along an arbitrary path. Such an approach is based on the principles of linear elastic fracture mechanics with the extended finite element method or with the adaptive remeshing technique. The onset and growth of the crack are also characterized by the fracture energy release rates or the stress intensity factors at the crack tip based on the Paris law (see *Paris, 1961*).

If both failure mechanisms (that is, discrete fatigue crack growth in the bulk brittle material and fatigue delamination growth at the brittle material interfaces) are considered within a single analysis, the most critical failure mechanism governs the actual fatigue crack growth and the damage in the region governed by the less critical failure mechanism is scaled proportionally. In the vicinity where fracture or debonding occurs, linear elastic deformation or the small scale yielding condition must be satisfied.

A fatigue crack growth analysis step can be the only step in an analysis, can follow a general or linear perturbation step, or can be followed by a general or linear perturbation step. Multiple fatigue crack growth analysis steps can be included in a single analysis except when used with the adaptive remeshing technique. The fatigue crack growth procedure supports only constant amplitude loading—thermal, mechanical, or a combination of thermal and mechanical. You must specify the cyclic loading amplitude curves for a single loading cycle. Such a general formulation allows a wide range of loading histories such as contact or complex combinations of asynchronous loadings within a cyclic loading definition. For example, a mechanical pressure and a temperature with peaks/troughs in each can occur at different times within a single loading cycle.

The crack growth is governed by the Paris law:

$$\frac{da}{dN} = c_3 \Delta G^{c_4},$$

where c_3 and c_4 are material constants.

An equivalent form of the above Paris law based on the stress intensity factor is also available for enriched elements or when used with the adaptive remeshing technique:

$$\frac{da}{dN} = c_3 \Delta K^{c_4}.$$

Ratcliffe and Johnston (2014) and *Deobald et al. (2017)* proposed the following alternate form of the fatigue law which better accounts for mixed-mode fatigue crack growth:

$$\frac{da}{dN} = c_3 G_{TMax}^{c_4}.$$

In the above expression, G_{TMax} is the total maximum strain energy release rate (as opposed to the strain energy release rate change over a cycle used in the original form of the Paris law), while c_3 and c_4, are material parameters that depend on mode mix and stress ratios. Abaqus does not support the above form of the crack growth rate equation directly, but instead allows specification of da/dN as a tabular function of G_{TMax}, the mode mix ratio, and the stress ratio.

The details of the usages of the different fatigue crack growth models are discussed in *PRX-Fatigue Criterion* for crack growth along initially partially bonded surfaces, in *Fatigue Crack Growth Criterion Based on the*

Principles of LEFM for crack growth in enriched elements, and in *Fatigue Crack Growth Criterion* for fatigue crack growth with adaptive remeshing.

In addition, a user-defined fatigue crack growth law can be specified in user subroutine *UMIXMODEFATIGUE*.

You specify the maximum numbers of cycles, N_{max}, when you define the fatigue crack growth analysis step.

Input File Usage: Use the following option to specify a general fatigue crack growth analysis with constant amplitude:

**FATIGUE*, TYPE=CONSTANT AMPLITUDE (default)
first data line
, , N_{max}

Simplifying the Fatigue Crack Growth Analysis

The general fatigue crack growth analysis procedure described above can be simplified in some special cases if the following conditions are satisfied:

- the peak or the trough value of the strain energy release rate, *G*, always occurs when the applied load, *P*, reaches its maximum or minimum value;
- the strain energy release rate is proportional to the square of the applied load, *P*; and
- the contact conditions remain unchanged during a single loading cycle.

For the simplified fatigue crack growth analysis, you can apply a constant load with a magnitude of $P_{\max}\sqrt{(1-\alpha^2)}$ (for the fracture energy release rate–based Paris law) or $P_{\max}(1-\alpha)$ (for the stress intensity factor–based Paris law), where $\alpha = P_{\min}/P_{\max}$, $P_{\max}$ is the maximum applied load and $P_{\min}$ is the minimum applied load over a single cycle. At least two increments are required for each single loading cycle period when the simplified method is used.

Input File Usage: Use the following option to apply a constant load to simplify the fatigue crack growth analysis:

**FATIGUE*, TYPE=SIMPLIFIED
first data line
, , N_{max}

Controlling the Incrementation during the Cyclic Time Period

Several automatic incrementation methods are available. Alternatively, you can use fixed time incrementation.

Automatic Incrementation

If you specify only the maximum allowable nodal temperature change in an increment, the time increments are selected automatically based on this value. Abaqus/Standard restricts the time increments to ensure that the maximum temperature change is not exceeded at any node during any increment of the analysis.

For rate-dependent constitutive equations you can limit the size of the time increment by the accuracy of the integration. The user-specified accuracy tolerance parameter limits the maximum inelastic strain rate change allowed over an increment:

$$tolerance \geq \left(\dot{\bar{\varepsilon}}^{cr}\big|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}\big|_{t}\right)\Delta t,$$

where

t

is the time at the beginning of the increment,

Δt

is the time increment (so that $t + \Delta t$ is the time at the end of the increment), and

$\dot{\bar{\varepsilon}}^{cr}$

is the equivalent creep strain rate.

To achieve sufficient accuracy, the value chosen for the accuracy tolerance parameter should be on the order of σ_{err}/E for creep problems (where σ_{err} is an acceptable level of error in the stress and E is a typical elastic modulus) or on the order of the elastic strains for viscoelasticity problems.

If rate-dependent constitutive equations are used in combination with a varying temperature, both controls can be used simultaneously. Abaqus/Standard chooses the increments that satisfy both criteria.

If neither the accuracy tolerance parameter nor the maximum allowable nodal temperature change is specified, Abaqus/Standard selects increment sizes based on computational efficiency.

Input File Usage: Use the following option to indicate the time increment sizes based on computational efficiency:

**FATIGUE*

Use the following option to specify the maximum allowable nodal temperature change:

**FATIGUE*, DELTMX=$\Delta\theta_{max}$

Use the following option to specify the accuracy tolerance parameter:

**FATIGUE*, CETOL=*tolerance*

Fixed Time Incrementation

If fixed time incrementation is preferred, you must specify the time increment, Δt, and the time period, T.

Input File Usage: **FATIGUE*, DIRECT
Δt, T

Defining the Time Points at Which the Response Must Be Evaluated

The user-defined time incrementation for a fatigue crack growth analysis step can be augmented or superseded by specifying particular time points in the loading history at which the response of the structure should be evaluated. This feature is particularly useful if you know prior to the analysis at which time points in the analysis the load reaches a maximum and/or minimum value or when the response will change rapidly. An example is the analysis of the heating/cooling thermal cycle of an engine component where you typically know when the temperature reaches a maximum value.

When time points are used with fixed time incrementation, the time incrementation specified for the fatigue crack growth step is ignored; instead, the time incrementation precisely follows the specified time points. If time

points are used with automatic incrementation, the time incrementation is variable; however, the response of the structure is evaluated at the specified time points.

The time points can be listed individually, or they can be generated automatically by specifying the starting time point, ending time point, and increment in time between the two specified time points.

Input File Usage: Use the following options to list the time points individually:

**TIME POINTS*, NAME=*time points name*
**FATIGUE*, TIME POINTS=*time points name*

Use the following options to generate the time points automatically:

**TIME POINTS*, NAME=*time points name*, GENERATE
**FATIGUE*, TIME POINTS=*time points name*

Discrete Crack Propagation along an Arbitrary Path with the Extended Finite Element Method

Fatigue crack growth analysis in Abaqus/Standard allows the modeling of discrete crack growth along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method. You complete the definition of the crack propagation capability by defining a fracture-based surface behavior and specifying the fracture criterion in enriched elements. The fracture energy release rates at the crack tips in enriched elements are calculated based on the modified virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue crack growth occurs, although nonlinear material deformations can occur somewhere else in the bulk materials. For more information about defining fracture criteria and VCCT in enriched elements, see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*.

To accelerate the fatigue crack growth analysis, the damage extrapolation technique is used, which advances the crack by at least one element length after each completed cycle.

Onset and Growth of Fatigue Crack

The onset and growth of fatigue crack at an enriched element are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue crack growth:

- one criterion is based on material constants, ΔG, and the current cycle number, N;
- the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value.

Once the onset of fatigue crack growth criterion is satisfied at the enriched elements, the crack growth rate, $\frac{da}{dN}$, is a piecewise function based on a user-specified form of the Paris law. The criteria for fatigue crack onset and growth are discussed in detail in *Fatigue Crack Growth Criterion Based on the Principles of LEFM*. If you do not specify the onset criterion, Abaqus/Standard assumes that the onset of fatigue crack growth is satisfied automatically.

Damage Extrapolation Technique

If the onset of the crack growth criterion is satisfied at any crack tip in the enriched element at the end of a completed cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by fracturing at least one enriched element ahead of the crack tips. Given the particular fatigue crack growth form of the Paris law (as defined in *Fatigue Crack Growth Criterion Based on the Principles*

of LEFM), combined with the known element length and likely propagation direction $\Delta a_{Nj} = a_{N+\Delta N} - a_N$ at the enriched elements ahead of the crack tips, the number of cycles necessary to fail each enriched element ahead of the crack tip can be calculated as ΔN_j, where j represents the enriched element ahead of the jth crack tip. The analysis is set up to advance the crack by at least one enriched element per increment after the loading cycle is completed. The element with the fewest cycles is identified to be fractured, and its $\Delta N_{min} = min\left(\Delta N_j\right)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{Nmin} = min\left(\Delta a_{Nj}\right)$. The most critical element is completely fractured with a zero constraint and a zero stiffness at the cracked surfaces at the end of the completed cycle. As the enriched element is fractured, the load is redistributed, and a new fracture energy release rate must be calculated for the enriched elements ahead of the crack tips for the next cycle. This capability allows at least one enriched element ahead of the crack tips to be fractured after each completed cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Fatigue Crack Growth with Adaptive Remeshing

Fatigue crack growth with adaptive remeshing is based on Abaqus fracture/fatigue technology in conjunction with tetrahedral element–based morphing and adaptive remeshing technology.

Abaqus/Standard calculates the stress intensity factors based on the conventional contour integral method (*Contour Integral Evaluation*). The crack growth is governed by the stress intensity factor or the strain energy release rate based on Paris' law. The tetrahedral element–based adaptive remeshing technology is used to create a new mesh after each increment of crack growth, based upon the updated crack front. The overall fatigue workflow is semiautomated and involves automatic execution (with no user intervention) of a series of successive Abaqus analyses with progressively longer crack lengths. No state mapping is performed or is necessary between two successive Abaqus analyses. For more information about defining fracture onset and growth criteria, controlling the incremental crack growth, and adaptive remeshing, see *Fatigue Crack Propagation with Adaptive Remeshing*.

Progressive Delamination Growth along a Predefined Path at Interfaces

Fatigue crack growth analysis in Abaqus/Standard also allows the modeling of progressive delamination growth at the interfaces in laminated composites. The interface along which the delamination (or crack) propagates must be indicated in the model using a fracture criterion definition. The fracture energy release rates at the crack tips in the interface elements are calculated based on the virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue delamination growth occurs along predefined surfaces, although nonlinear material deformations can occur in the bulk materials. For more information about defining fracture criteria and VCCT, see *Crack Propagation Analysis*.

To accelerate the fatigue crack growth analysis, the damage extrapolation technique is used, which releases at least one element length at the crack tip along the interface after each completed cycle. When both brittle fatigue delamination at interfaces and discrete crack growth in bulk materials are considered in an analysis, failure occurs first at the weakest link.

Onset and Growth of Fatigue Delamination

The onset and growth of fatigue delamination at a defined crack interface are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue delamination growth:

- one criterion is based on material constants, ΔG, and the current cycle number, N;
- the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value.

Once the onset of the delamination growth criterion is satisfied at the interface, the delamination growth rate, $\frac{da}{dN}$, is a piecewise function based on a user-specified form of the Paris law. The criteria for fatigue delamination onset and growth are discussed in detail in *Fatigue Crack Growth Criterion*. If you do not specify the onset criterion, Abaqus/Standard assumes that the onset of fatigue crack growth is satisfied automatically.

Damage Extrapolation Technique at the Interface Elements

If the onset of delamination growth criterion is satisfied at any crack tip in the interface at the end of a completed cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by releasing at least one element at the interface. Given the particular fatigue crack growth form of the Paris law (as defined in *Fatigue Crack Growth Criterion*), combined with the known node spacing $\Delta a_{Nj} = a_{N+\Delta N} - a_N$ at the interface elements at the crack tips, the number of cycles necessary to fail each interface element at the crack tip can be calculated as ΔN_j, where *j* represents the node at the *j*th crack tip. The analysis is set up to release at least one interface element per increment after the loading cycle is completed. The element with the fewest cycles is identified to be released, and its $\Delta N_{min} = min\left(\Delta N_j\right)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{Nmin} = min\left(\Delta a_{Nj}\right)$. The most critical element is completely released with a zero constraint and a zero stiffness at the end of the completed cycle. As the interface element is released, the load is redistributed, and a new relative fracture energy release rate must be calculated for the interface elements at the crack tips for the next cycle. This capability allows at least one interface element at the crack tips to be released after each completed cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Controlling Element Fracture

In the following discussion, the terms damage, fracture, and debonding are used in a generic sense to describe the process by which the internal forces at the crack-tip nodes are reduced to zero as they transition from an uncracked to a fully cracked state. At the end of each cycle, Abaqus utilizes the stress state at the nodes of the current crack front to determine the state of damage at each node. The following fundamental assumptions govern the fracture process:

- The most critical element ahead of the current crack front undergoes complete damage.
- The total cycle count increases by ΔN_{min}, the number of cycles (based on the Paris law) needed to fracture the most critical element.
- In addition to elements forecast to be fully or almost fully damaged after ΔN_{min} cycles, additional elements are also allowed to partially fracture if they are within certain tolerances (described below) in the current cycle.

Allowing partial damage in some elements helps ensure a relatively smooth (that is, nonjagged) new crack front. The traction at a crack-tip node is either removed immediately upon fracture or ramped down gradually (see *Specifying How a Debonding Force Is Released after a Fracture Criterion Is Met in Abaqus/Standard*).

For the discussion that follows, it is convenient to introduce a damage variable that is defined in an "incremental" (defined in terms of cycle increments) framework as:

$$D_{N+\Delta N} = D_N + \Delta D,$$

with

$$\Delta D = \frac{\frac{da}{dN}\Delta N_{\min}}{a_N},$$

where

$D_{N+\Delta N}$ and D_N

are the scalar damage variables at the end of cycles $N + \Delta N$ and N, respectively, and

a_N

is the length of the element ahead of a current crack front node.

The value of D varies between 0 (undamaged state) and 1 (fully damaged state), with intermediate values indicating partial damage at a crack-tip node. At the beginning of the analysis ($N = 0$), the initial damage at each crack-tip node is $D_0 = 0$. At the end of the first completed cycle increment, ΔN, the crack-tip node for the most critical element satisfies the condition $\Delta D = 1$, and, hence, is fully released.

For other crack-tip nodes, $0 \le \Delta D < 1$, and these nodes are either undamaged or only partially damaged. The partial damage is governed by the damage variable, D. Abaqus reduces the effective length of partially damaged elements as:

$$\Delta a_N^{eff} = \left(1 - D\right)\Delta a_N.$$

The reduced effective length is used for the next crack growth calculations based on the Paris law.

Two criteria are available to control the partial fracture of the elements ahead of the current crack front:

- A cycle-based criterion that is based on a tolerance value, ΔD_{Ntol}.
- A damage-based criterion that is based on a tolerance value, ΔD_{Dtol}.

If neither tolerance is specified, the damage-based criterion, with a default $\Delta D_{Dtol} = 0.25$, is assumed to be in effect. If both tolerances are specified, the damage-based tolerance takes precedence. Using a damage-based tolerance allows you to choose the tolerance independent of the cycle increment size during the analysis.

In addition to satisfying one of the two criteria outlined above, crack front nodes are only fractured partially if such fracture helps ensure overall self-similar crack propagation, which is a fundamental assumption of the VCCT method. In other words, not all nodes that are eligible for partial fracture based on either the cycle-based or the damage-based criterion actually undergo partial fracture.

Cycle-Based Criterion for Partial Damage

For a cycle-based criterion, all elements ahead of the current crack front that satisfy the following criterion are eligible for partial fracture:

$$\frac{Log\Delta N_j - Log\Delta N_{min}}{Log\Delta N_{min}} \le \Delta D_{Ntol}.$$

A typical value of ΔD_{Ntol} that provides a balance between accuracy and performance is 0.1.

Input File Usage: Use the following option to specify a cycle-based tolerance for determining the elements that are eligible for partial fracture:

**FATIGUE*
first data line
, , , , ΔD_{Ntol},

Damage-Based Criterion for Partial Damage

For a damage-based criterion, all elements ahead of the current crack front that satisfy the following criterion are eligible for partial fracture:

$$D_{N+\Delta N} \geq 1 - \Delta D_{Dtol}.$$

Thus, for the default choice of $\Delta D_{Dtol} = 0.25$, all elements ahead of crack tip nodes where $D_{N+\Delta N} \geq 0.75$ are partially released.

Input File Usage: Use the following option to specify a damage-based tolerance for determining the elements that are eligible for partial fracture:

**FATIGUE*
first data line
, , , , , ΔD_{Dtol}

Influence of Tolerance Parameters on Accuracy and Performance

If a small tolerance is specified, fewer elements along the crack front are eligible for fracture during any given cycle increment ΔN. In this case, more cycle increments in total (with a smaller size for each cycle increment) are required to propagate an initial crack to its final allowable crack length. This approach results in a more accurate solution, although at a higher computational cost.

The opposite is true if a large tolerance is specified. In this case, more elements along the crack front are eligible for fracture during any given cycle increment. Therefore, fewer cycle increments in total (with a larger size for each cycle increment) are likely required to propagate an initial crack to its final allowable crack length. This approach might be computationally more efficient. However, the predicted total number of cycles taken to propagate an initial crack to its final allowable crack size might be less conservative. In addition, the use of a very large tolerance (for example, 0.95) might violate the self-similar crack growth assumption that is fundamental to the VCCT method, resulting in a jagged (not smoothing) crack front.

Figure 1 illustrates how the change of tolerance influences the predicted response of cycle number versus crack length in a mixed-mode fatigue crack growth analysis when compared to a benchmark solution (*Krueger et al, 2020*). Reducing the tolerance from 0.25 to 0.1 results in an almost identical response; however, the solution with a tolerance of 0.1 was computationally more expensive. For a larger tolerance, the predicted total number of cycles taken to propagate an initial crack to its final allowable crack length is somewhat nonconservative.

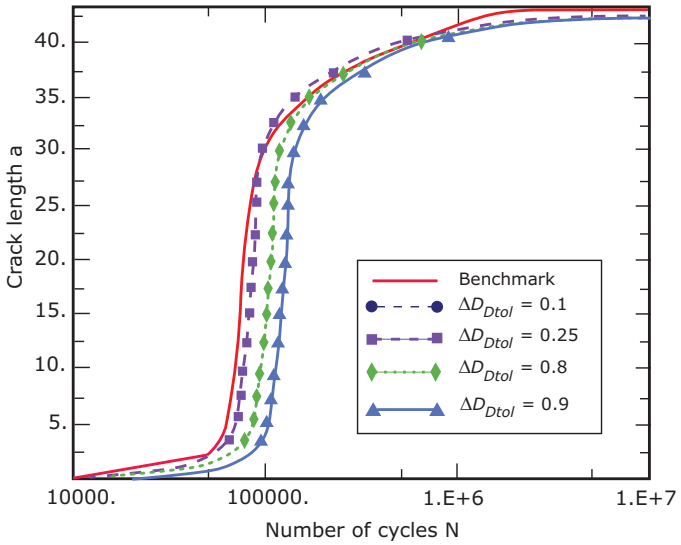


Figure 1: Effect of the tolerance on the predicted response of cycle number versus crack length.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see *Initial Conditions*).

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom. During the analysis, prescribed boundary conditions in a general fatigue crack growth step must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). However, prescribed boundary conditions in a simplified fatigue crack growth analysis must have a constant value. If the analysis consists of several steps, the usual rules apply (see *Boundary Conditions*). At each new step, the boundary condition can either be modified or completely defined. All boundary conditions defined in previous steps remain unchanged unless they are redefined.

Loads

The following loads can be prescribed in a fatigue crack growth analysis step:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in the *Abaqus Elements Guide*.

During the general fatigue crack growth analysis, each load must have an amplitude definition that is cyclic over the step where the start value must be equal to the end value (see *Amplitude Curves*). However, each load must have a constant value in a simplified fatigue crack growth analysis. If the analysis consists of several steps, the usual rules apply (see *About Loads*). At each new step, the loading can either be modified or completely defined. All loads defined in previous steps remain unchanged unless they are redefined.

Predefined Fields

The following predefined fields can be specified in a fatigue crack growth analysis step, as described in *Predefined Fields*:

- Temperature is not a degree of freedom in a fatigue crack growth analysis step, but nodal temperatures can be specified as a predefined field. The temperature values specified in a general fatigue crack growth analysis must be cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see *Initial Conditions*). Alternatively, you can ramp the temperatures back to their initial condition values, as described in *Predefined Fields*. Any difference between the applied and initial temperatures causes thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any. In a simplified fatigue crack growth analysis, the temperature values specified must be constant.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. The field variable values specified in a general fatigue crack growth analysis must be cyclic over the step. The field variable values must be constant in a simplified fatigue crack growth analysis step.

Material Options

Most material models that describe mechanical behavior are available for use in a fatigue crack growth analysis. The inelastic definition in a material point must be used in conjunction with the linear elastic material model (*Linear Elastic Behavior*), the porous elastic material model (*Elastic Behavior of Porous Materials*), or the hypoelastic material model (*Hypoelastic Behavior*).

The following material properties are not active during a fatigue crack growth analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, piezoelectric properties, and pore fluid flow properties.

Rate-dependent yield (*Rate-Dependent Yield*), rate-dependent creep (*Rate-Dependent Plasticity: Creep and Swelling*), and two-layer viscoplasticity (*Two-Layer Viscoplasticity*) can also be used during a fatigue crack growth analysis.

However, in the vicinity where fracture or debonding occurs, linear elastic deformation or the small scale yielding condition must be satisfied.

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a fatigue crack growth analysis (see *Choosing the Appropriate Element for an Analysis Type*). However, when modeling fatigue crack growth based on the principles of linear elastic fracture mechanics with the extended finite element method, only first-order continuum stress/displacement elements and second-order stress/displacement tetrahedral elements can be associated with an enriched feature (see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*).

When modeling fatigue crack growth with the adaptive remeshing technique (see *Fatigue Crack Propagation with Adaptive Remeshing*), only second-order tetrahedral stress-displacement continuum elements can be used in the adaptive subdomain. You may use any of the stress-displacement elements available in the Abaqus/Standard element library in the non-adaptive subdomain (see *Choosing the Appropriate Element for an Analysis Type*).

Output

In addition to the standard output identifiers available in Abaqus (*Abaqus/Standard Output Variable Identifiers*), whole element and surface variables are available.

The following whole element variables are available with the extended finite element method:

STATUSXFEM Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked, 0.0 if the element is not. If the element is partially cracked, the value lies between 1.0 and 0.0.)

CYCLEINIXFEM Number of cycles to initialize the crack at the enriched element.

CYCLEXFEM Number of cycles to fracture at the enriched element.

ENRRTXFEM All components of strain energy release rate.

The following additional surface output variables can be also requested along a predefined path at interfaces:

CSDMG Overall value of the scalar damage variable.

BDSTAT	Bond state. The bond state varies between 1.0 (fully bonded) and 0.0 (fully unbonded).
CYCLE	Number of cycles to debond.
ENRRT	All components of strain energy release rate.
CRKLENGTH	Accumulated crack length. It is measured starting from the unbonded nodes immediately behind the initial crack front. As the crack tip nodes debond, the debonded element length is added to the accumulated crack length.

Limitations

The fatigue crack growth procedure supports only constant amplitude loading—thermal, mechanical, or a combination of thermal and mechanical. Several fatigue crack growth analysis steps can be used for an analysis with variable amplitude loading with each step having a constant amplitude loading.

Significant inaccuracy in fatigue prediction can occur if the fatigue procedure is used for cases that depart significantly from linear elastic response near a crack. See *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach* for discussion of simulation fatigue crack growth involving ductile materials.

Input File Template

The following is an example using the general fatigue crack growth analysis procedure:

```
*HEADING
…
*ENRICHMENT, TYPE=PROPAGATION CRACK, INTERACTION=INTERACTION,
ELSET=ENRICHED
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*MATERIAL
Options to define material properties
*SURFACE, INTERACTION=INTERACTION
*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in the Paris law and  fracture criterion in the bulk
material for enriched elements
**
*SURFACE, NAME=secondary
Data lines to define the secondary surface at the delamination interface
*SURFACE, NAME=main
Data lines to define the main surface at the delamination interface
*CONTACT PAIR
secondary, main
*TIME POINTS, NAME=T1
**
*STEP (,INC=)
Set  INC equal to the maximum number of increments in a single loading cycle
```

```
*FATIGUE, TYPE=CONSTANT AMPLITUDE, TIME POINTS=T1
Data line to define time increment, cycle time, minimum time increment allowed, and maximum time increment
 allowed
Data line to define minimum increment in number of cycles, maximum increment in number of cycles, total
number of cycles, , tolerance for the least number of cycles to fracture
*DEBOND, SECONDARY=secondary, MAIN=main
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in the Paris law and  fracture criterion at the interface
**
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
**
*END STEP
```

The following is an example using the simplified fatigue crack growth analysis procedure:

```
*HEADING
…
*ENRICHMENT, TYPE=PROPAGATION CRACK, INTERACTION=INTERACTION,
ELSET=ENRICHED
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define a constant load equal to P_max sqrt(1 − α^2)
**
*MATERIAL
Options to define material properties
*SURFACE, INTERACTION=INTERACTION
*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in the Paris law and  fracture criterion in the bulk
material for enriched elements
**
*SURFACE, NAME=secondary
Data lines to define the secondary surface at the delamination interface
*SURFACE, NAME=main
Data lines to define the main surface at the delamination interface
*CONTACT PAIR
secondary, main
*TIME POINTS, NAME=T1
**
*STEP (,INC=)
Set  INC equal to the maximum number of increments in a single loading cycle (at least two increments are
required)
*FATIGUE, TYPE=SIMPLIFIED, TIME POINTS=T1
Data line to define time increment, cycle time, minimum time increment allowed, and maximum time increment
 allowed
Data line to define minimum increment in number of cycles, maximum increment in number of cycles, total
number of cycles, , tolerance for the least number of cycles to fracture
*DEBOND, SECONDARY=secondary, MAIN=main
*FRACTURE CRITERION, TYPE=FATIGUE
```

```
Data lines to define material constants used in the Paris law and  fracture criterion at the interface
**
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
**
*END STEP
```

References

- Deobald, L., G. Mabson, S. Engelstad, M. Rao, M. Gurvich, W. Seneviratne, S. Perera, T. O'Brien, G. Muri, J. Ratcliffe, C. Davila, N. Carvalho, and R. Krueger, "Guidelines for VCCT-Based Interlaminar Fatigue and Progressive Failure Finite Element Analysis," *NASA/TM-2017-219663*, 2017.
- Krueger, R., L. Deobald, and H. Gu, "A Benchmark Example for Delamination Growth Predictions Based on the Single Leg Bending Specimen under Fatigue Loading," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 7, no. 11, 2020.
- Paris, P., M. Gomaz, and W. Anderson, "A Rational Analytic Theory of Fatigue," *TheTrend in Engineering*, vol. 15, 1961.
- Ratcliffe, J., and W. Johnston, "Influence of Mixed Mode I-Mode II Loading on Fatigue Delamination Growth Characteristics of a Graphite Epoxy Tape Laminate," *Proceedings of American Society for Composites 29th Technical Conference*, 2014.

Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach

Products: Abaqus/Standard Abaqus/CAE

Overview

A low-cycle fatigue analysis:

- is characterized by states of stress high enough for inelastic deformation to occur in most cases;
- is a quasi-static analysis on a structure subjected to sub-critical cyclic loading;
- can be associated with thermal as well as mechanical loading;
- uses the direct cyclic approach to obtain the stabilized cyclic response of the structure directly;
- models progressive damage and failure in bulk ductile material based on a continuum damage mechanics approach, in which case damage initiation and evolution are characterized by the accumulated inelastic hysteresis strain energy per stabilized cycle;
- models propagation of a discrete crack along an arbitrary, solution-dependent path without remeshing in the bulk brittle material based on the principles of linear elastic fracture mechanics (LEFM) with the extended finite element method, in which case the onset and growth of fatigue crack are characterized by the relative fracture energy release rate;
- models progressive delamination growth along a predefined path at the brittle material interfaces in laminated composites, in which case the onset and growth of fatigue delamination at the interfaces are characterized by the relative fracture energy release rate;
- uses the damage extrapolation technique to accelerate the low-cycle fatigue analysis; and
- assumes geometrically linear behavior and fixed contact conditions within each loading cycle.

In simulations where the bulk material deformation is inelastic, the direct cyclic approach is the preferred method. It can be much more computationally efficient at obtaining a stabilized response than a classical transient analysis, which might require the application of many loading cycles to obtain the same result. However, in the case of linear elastic response with brittle materials, it might not be optimal, or even desirable, to use a Fourier series to represent the displacement and residual fields. The preferred method in this case is to use the classical incremental method (see *Linear Elastic Fatigue Crack Growth Analysis*).

References:

- *Defining an Analysis*
- *About Static Stress Analysis Procedures*
- *Direct Cyclic Analysis*
- *Linear Elastic Fatigue Crack Growth Analysis*
- *Crack Propagation Analysis*
- *Damage and Failure for Ductile Materials in Low-Cycle Fatigue Analysis*
- *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*
- **DAMAGE EVOLUTION*
- **DAMAGE INITIATION*
- **DEBOND*
- **DIRECT CYCLIC*
- **FRACTURE CRITERION*

- **CONTROLS*
- *Configuring a direct cyclic procedure*

Approaches to Low-Cycle Fatigue Analysis

The traditional approach for determining the fatigue limit for a structure is to establish the $S - N$ curves (load versus number of cycles to failure) for the materials in the structure. Such an approach is still used as a design tool in many cases to predict fatigue resistance of engineering structures. However, this technique is generally conservative, and it does not define a relationship between the cycle number and the degree of damage or crack length.

One alternative approach is to predict the fatigue life by using a crack/damage evolution law based on the inelastic strain/energy when the structure's response is stabilized after many cycles. Because the computational cost to simulate the slow progressive damage in a material over many load cycles is prohibitively expensive for all but the simplest models, numerical fatigue life studies usually involve modeling the response of the structure subjected to a small fraction of the actual loading history. This response is then extrapolated over many load cycles using empirical formulas such as the Coffin-Manson relationship (see *Coffin, 1954,* and *Manson, 1953*) to predict the likelihood of crack initiation and propagation. Since this approach is based on a constant crack/damage growth rate, it might not realistically predict the evolution of the crack or damage.

Low-Cycle Fatigue Analysis in Abaqus/Standard

The direct cyclic analysis capability in Abaqus/Standard provides a computationally effective modeling technique to obtain the stabilized response of a structure subjected to periodic loading and is ideally suited to perform low-cycle fatigue calculations on a large structure. The capability uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized response of the structure directly. The theory and algorithm to obtain a stabilized response using the direct cyclic approach are described in detail in *Direct cyclic algorithm*.

The direct cyclic low-cycle fatigue procedure models the progressive damage and failure both in bulk materials (such as in solder joints in an electronic chip packaging or intra-laminar crack growth in laminated composites) and at material interfaces (such as delamination in laminated composites). The former can be based on either a continuum damage mechanics approach or the principles of linear elastic fracture mechanics with the extended finite element method. The response is obtained by evaluating the behavior of the structure at discrete points along the loading history (see *Figure 1*). The solution at each of these points is used to predict the degradation and evolution of material properties that will take place during the next increment, which spans a number of load cycles, ΔN. The degraded material properties are then used to compute the solution at the next increment in the load history. Therefore, the crack/damage growth rate is updated continually throughout the analysis.

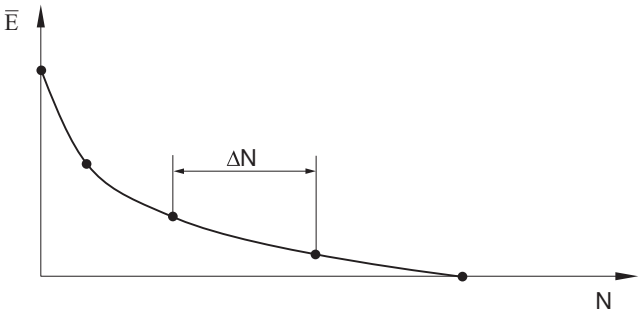


Figure 1: Elastic stiffness degradation as a function of the cycle number.

The elastic material stiffness at a material point remains constant and contact conditions remain unchanged when the stabilized solution is computed at a given point in the loading history. Each of the solutions along the loading history represents the stabilized response of the structure subjected to the applied period loads, with a level of

material damage at each point in the structure computed from the previous solution. This process is repeated up to a point in the loading history at which a fatigue life assessment can be made.

In bulk material, there are two approaches to modeling the progressive damage and failure. One approach is based on continuum damage mechanics. This approach is more appropriate for ductile material, in which the cyclic loading leads to stress reversals and the accumulation of plastic strains, which in turn cause the initiation and propagation of cracks. The damage initiation and evolution are characterized by the stabilized accumulated inelastic hysteresis strain energy per cycle as illustrated in *Figure 2*. The other approach is based on the principles of linear elastic fracture mechanics with the extended finite element method. This approach is more appropriate for brittle material or material with small scale yielding, in which the cyclic loading leads to material strength degradation causing fatigue crack growth along an arbitrary path. The onset and growth of the crack are characterized by the relative fracture energy release rate at the crack tip based on the Paris law (*Paris, 1961*).

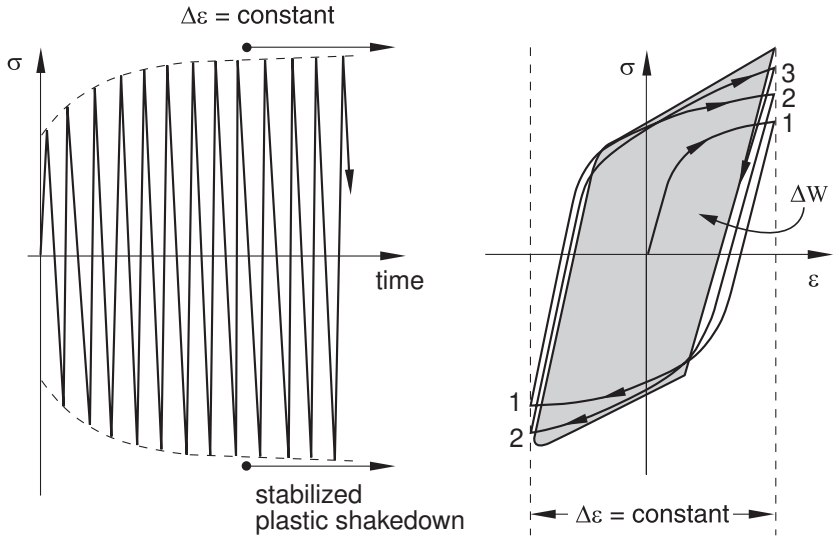


Figure 2: Plastic shakedown in a direct cyclic analysis.

At interfaces of laminated composites, the cyclic loading leads to interface strength degradation causing fatigue delamination growth. The onset and growth of delamination are also characterized by the relative fracture energy release rate at the crack tip based on the Paris law (*Paris, 1961*).

Both the progressive damage mechanism in the bulk material and the progressive delamination growth mechanism at interfaces can be considered simultaneously, with the failure occurring first at the weakest link in a model.

Defining a low-cycle fatigue analysis using the direct cyclic approach is similar to defining a direct cyclic analysis. See *Direct Cyclic Analysis* for details on how to specify the number of Fourier terms, number of iterations, and the increment sizes. You specify the maximum numbers of cycles, N_{max}, when you define the low-cycle fatigue analysis step.

Input File Usage: **DIRECT CYCLIC*, FATIGUE
first data line
, , N_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Maximum number of cycles: Value:** N_{max}

Determining Whether to Use the Fourier Coefficients from the Previous Step

A low-cycle fatigue step using the direct cyclic approach can be the only step in an analysis, can follow a general or linear perturbation step, or can be followed by a general or linear perturbation step. Multiple low-cycle fatigue analysis steps can be included in a single analysis. In such a case the Fourier series coefficients obtained in the previous step can be used as starting values in the current step. By default, the Fourier coefficients are reset to zero, thus allowing application of cyclic loading conditions that are very different from those defined in the previous low-cycle fatigue step.

As in a direct cyclic analysis, you can specify that a low-cycle fatigue step in a restart analysis should use the Fourier coefficients from the previous step, thus allowing continuation of an analysis to simulate more loading cycles. In a low-cycle fatigue analysis, a restart file is written at the end of the stabilized cycle. Consequently, a restart analysis that is a continuation of a previous low-cycle fatigue analysis will start with a new loading cycle at $t = 0$ (see *Restarting an Analysis*).

Input File Usage: Use the following option to specify that the current step is a continuation of the previous low-cycle fatigue step using the direct cyclic approach:

**DIRECT CYCLIC*, FATIGUE, CONTINUE=YES

Use the following option to reset the Fourier series coefficients to zero:

**DIRECT CYCLIC*, FATIGUE, CONTINUE=NO (default)

Abaqus/CAE Usage: Use the following option to specify that the current step is a continuation of the previous low-cycle fatigue step using the direct cyclic approach:

Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Use displacement Fourier coefficients from previous direct cyclic step**; **Fatigue: Include low-cycle fatigue analysis**

Use the following option to reset the Fourier series coefficients to zero:

Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**

Progressive Damage and Damage Extrapolation in Bulk Ductile Material Based on Continuum Damage Mechanics Approach

Low-cycle fatigue analysis in Abaqus/Standard allows modeling of progressive damage and failure for ductile materials in any elements whose response is defined in terms of a continuum-based constitutive model (*About the Material Library*). This includes cohesive elements modeled using a continuum approach (*Modeling of an Adhesive Layer of Finite Thickness*). The inelastic definition in a material point must be used in conjunction with the linear elastic material model (*Linear Elastic Behavior*), the porous elastic material model (*Elastic Behavior of Porous Materials*), or the hypoelastic material model (*Hypoelastic Behavior*).

After damage initiation, the elastic material stiffness is degraded progressively in each cycle (as shown in *Figure 1*) based on the accumulated stabilized inelastic hysteresis energy. It is impractical and computationally expensive to perform a cycle-by-cycle simulation for a low-cycle fatigue analysis; Instead, to accelerate the low-cycle fatigue analysis, each increment extrapolates the current damaged state in the bulk material forward over many cycles to a new damaged state after the current loading cycle is stabilized.

Damage Initiation and Evolution

Damage initiation refers to the beginning of degradation of the response of a material point. In a low-cycle fatigue analysis the damage initiation criterion is characterized by the accumulated inelastic hysteresis energy per cycle, Δw. Δw and material constants are used to determine the number of the cycle in which damage is initiated, N_0. At the end of a stabilized loading cycle, N, Abaqus/Standard checks to see if the damage initiation criterion $N > N_0$ is satisfied in any material point; material stiffness at a material point will not be degraded unless this criterion is satisfied. The calculations and output associated with damage initiation are discussed in detail in *Damage Initiation for Ductile Materials in Low-Cycle Fatigue*.

Once the damage initiation criterion is satisfied at a material point, the damage state is calculated and updated based on the inelastic hysteresis energy for the stabilized cycle. Abaqus/Standard assumes that the degradation of the elastic stiffness can be modeled using the scalar damage variable, D. The rate of the damage in a material point per cycle, $\frac{dD}{dN}$, is calculated based on the accumulated inelastic hysteresis energy, the characteristic length associated with an integration point, and material constants. For details, see *Damage Evolution for Ductile Materials in Low-Cycle Fatigue*.

Typically, a material has completely lost its load-carrying capacity when $D = 1$. You can remove an element from the mesh if all the section points at all integration locations of the element have lost their load-carrying capability.

Damage Extrapolation Technique in the Bulk Material

If the damage initiation criterion is satisfied in any material point at the end of a stabilized cycle, N, Abaqus/Standard extrapolates the damage variable D_N from the current cycle forward to the next increment over a number of cycles, ΔN. The new damage state, $D_{N+\Delta N}$, is given by

$$D_{N+\Delta N} = D_N + \frac{\Delta N}{L} c_3 \left(\frac{\Delta w}{\Delta \bar{w}_0} \right)^{c_4},$$

where L is the characteristic length associated with an integration point, and c_3 and c_4 are material constants, and $\Delta \bar{w}_0$ is a reference value of the accumulated inelastic hysteresis energy density per cycle (see *Damage Evolution for Ductile Materials in Low-Cycle Fatigue* for more information).

You specify the minimum (ΔN_{min}) and maximum (ΔN_{max}) number of cycles over which the damage is extrapolated forward in any given increment. The default values are 100 and 1000, respectively.

Input File Usage: **DIRECT CYCLIC*, FATIGUE
first data line
ΔN_{min}, ΔN_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Cycle increment size: Minimum:** ΔN_{min}, **Maximum:** ΔN_{max}

Discrete Crack Propagation along an Arbitrary Path Based on the Principles of Linear Elastic Fracture Mechanics with the Extended Finite Element Method

Low-cycle fatigue analysis in Abaqus/Standard allows the modeling of discrete crack growth along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method. You complete the definition of the crack propagation capability by defining a fracture-based surface behavior and

specifying the fracture criterion in enriched elements. The fracture energy release rates at the crack tips in enriched elements are calculated based on the modified virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue crack growth occurs, although nonlinear material deformations can occur somewhere else in the bulk materials. For more information about defining fracture criteria and VCCT in enriched elements, see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*.

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used, which advances the crack by at least one element length after each stabilized cycle.

Onset and Growth of Fatigue Crack

The onset and growth of fatigue crack at an enriched element are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue crack growth: one criterion is based on material constants, ΔG, and the current cycle number, N; the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value. Once the onset of fatigue crack growth criterion is satisfied at the enriched elements, the crack growth rate, $\frac{da}{dN}$, is a piecewise function based on material constants and ΔG (the Paris law). The criteria for fatigue crack onset and growth are discussed in detail in *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*.

Damage Extrapolation Technique

If the onset of crack growth criterion is satisfied at any crack tip in the enriched element at the end of a stabilized cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by fracturing at least one enriched element ahead of the crack tips. Given the material constants c_3 and c_4 (as defined in *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*), combined with the known element length and likely propagation direction $\Delta a_{Nj} = a_{N+\Delta N} - a_N$ at the enriched elements ahead of the crack tips, the number of cycles necessary to fail each enriched element ahead of the crack tip can be calculated as ΔN_j, where j represents the enriched element ahead of the jth crack tip. The analysis is set up to advance the crack by at least one enriched element per increment after the loading cycle is stabilized. The element with the fewest cycles is identified to be fractured, and its $\Delta N_{min} = min\left(\Delta N_j\right)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{Nmin} = min\left(\Delta a_{Nj}\right)$. The most critical element is completely fractured with a zero constraint and a zero stiffness at the cracked surfaces at the end of the stabilized cycle. As the enriched element is fractured, the load is redistributed, and a new relative fracture energy release rate must be calculated for the enriched elements ahead of the crack tips for the next cycle. This capability allows at least one enriched element ahead of the crack tips to be fractured after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Progressive Delamination Growth along a Pre-Defined Path at Interfaces

Low-cycle fatigue analysis in Abaqus/Standard also allows the modeling of progressive delamination growth at the interfaces in laminated composites. The interface along which the delamination (or crack) propagates must be indicated in the model using a fracture criterion definition. The fracture energy release rates at the crack tips in the interface elements are calculated based on the virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue delamination growth occurs along predefined surfaces, although nonlinear material deformations can occur in the bulk materials. For more information about defining fracture criteria and VCCT, see *Crack Propagation Analysis*.

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used, which releases at least one element length at the crack tip along the interface after each stabilized cycle. When both brittle fatigue delamination at interfaces and ductile damage or discrete crack growth in bulk materials are considered in an analysis, failure occurs first at the weakest link.

Onset and Growth of Fatigue Delamination

The onset and growth of fatigue delamination at a defined crack interface are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue delamination growth: one criterion is based on material constants, ΔG, and the current cycle number, N; the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value. Once the onset of delamination growth criterion is satisfied at the interface, the delamination growth rate, $\frac{da}{dN}$, is a piecewise function based on material constants and ΔG (the Paris law). The criteria for fatigue delamination onset and growth are discussed in detail in *Fatigue Crack Growth Criterion*.

Damage Extrapolation Technique at the Interface Elements

If the onset of delamination growth criterion is satisfied at any crack tip in the interface at the end of a stabilized cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by releasing at least one element at the interface. Given the material constants c_3 and c_4 (as defined in *Fatigue Crack Growth Criterion*), combined with the known node spacing $\Delta a_{Nj} = a_{N+\Delta N} - a_N$ at the interface elements at the crack tips, the number of cycles necessary to fail each interface element at the crack tip can be calculated as ΔN_j, where *j* represents the node at the *j*th crack tip. The analysis is set up to release at least one interface element per increment after the loading cycle is stabilized. The element with the fewest cycles is identified to be released, and its $\Delta N_{min} = min\left(\Delta N_j\right)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{Nmin} = min\left(\Delta a_{Nj}\right)$. The most critical element is completely released with a zero constraint and a zero stiffness at the end of the stabilized cycle. As the interface element is released, the load is redistributed, and a new relative fracture energy release rate must be calculated for the interface elements at the crack tips for the next cycle. This capability allows at least one interface element at the crack tips to be released after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Controlling the Solution Accuracy

Low-cycle fatigue analysis utilizes the direct cyclic approach to obtain the stabilized cyclic solution iteratively by combining a Fourier series approximation with time integration of the nonlinear material behavior using a modified Newton method. The accuracy of the algorithm depends on the number of Fourier terms used, the number of iterations taken to obtain the stabilized solution, and the number of time points within the load period at which the material response and residual vector are evaluated. Some methods for controlling the solution accuracy in a direct cyclic analysis are described in detail in *Direct Cyclic Analysis*. They all remain valid in a low-cycle fatigue analysis using the direct cyclic approach. In addition, the accuracy of a low-cycle fatigue analysis depends on the number of cycles over which the damage is extrapolated forward, as described below.

Controlling the Accuracy of Damage Extrapolation in the Bulk Material When Using the Continuum Damage Mechanics Approach

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used at the end of a stabilized cycle. In addition to specifying the minimum and maximum number of cycles over which the damage is extrapolated (see *Damage Extrapolation Technique in the Bulk Material* above), you can specify the damage

extrapolation tolerance, ΔD_{tol}, to control the accuracy of damage extrapolation in the bulk material. The default is $\Delta D_{tol} = 1.0$.

Input File Usage: Use the following option to specify the damage extrapolation tolerance:

**DIRECT CYCLIC*, FATIGUE
first data line
, , , ΔD_{tol}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Damage extrapolation tolerance:** ΔD_{tol}

Determining the Increment over Which Damage Is Extrapolated Forward

Abaqus/Standard uses an adaptive algorithm to determine the number of cycles over which the damage is extrapolated forward in each increment. By default, Abaqus/Standard starts with 500 cycles (half of the default value of maximum increment in number of cycles) and determines the maximum damage increment at any material points based on

$$\Delta D = \frac{\Delta N c_3 \left(\frac{\Delta w}{\Delta \overline{w}_0} \right)^{c_4}}{L} < \Delta D_{tol}.$$

If the maximum damage increment, ΔD, is greater than the damage extrapolation tolerance that you specify, the number of cycles over which the damage is extrapolated forward is reduced accordingly to ensure the maximum damage increment is less than the damage extrapolation tolerance. On the other hand, if the maximum damage increment at all material points is less than half of the damage extrapolation tolerance that you specify, the number of cycles is increased accordingly to ensure the maximum damage increment is equal to the damage extrapolation tolerance.

Controlling Element Fracture

In addition to elements forecast to be fully or almost fully damaged after ΔN_{min}, additional elements are allowed to fracture if they are within the tolerances described below in the current cycle. This approach avoids a jagged (not smooth) crack front. The traction is removed immediately on fracture or ramped down gradually (see *Specifying How a Debonding Force Is Released after a Fracture Criterion Is Met in Abaqus/Standard*).

Two criteria are available to control additional fracture of elements ahead of the current crack front: a cycle-based criterion (with a tolerance ΔD_{Ntol}) and a damage-based criterion (with a tolerance ΔD_{Dtol}). If both tolerances are specified, the damage-based tolerance takes precedence.

Elements that satisfy the following expression fracture if the cycle-based criterion is in effect:

$$\frac{Log\Delta N_j - Log\Delta N_{min}}{Log\Delta N_{min}} \leq \Delta D_{Ntol}.$$

Elements that satisfy the following expression fracture if the damage-based criterion is in effect:

$$D_{N+\Delta N} = D_N + \frac{\frac{da}{dN}\Delta N_{\min}}{a_N} \geq 1 - \Delta D_{Dtol},$$

where $D_{N+\Delta N}$ and D_N are the scalar damage variables at the end of cycles $N + \Delta N$ and N, respectively.

Input File Usage: Use the following option to specify the tolerance for the least number of cycles to fracture an element:

**DIRECT CYCLIC*, FATIGUE
first data line
, , , , ΔD_{Ntol}, ΔD_{Dtol}

Abaqus/CAE Usage: Specifying a tolerance for the least number of cycles to fracture an element is not supported in Abaqus/CAE.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see *Initial Conditions*).

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom. During the analysis, prescribed boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the analysis consists of several steps, the usual rules apply (see *Boundary Conditions*). At each new step, the boundary condition can either be modified or completely defined. All boundary conditions defined in previous steps remain unchanged unless they are redefined.

Loads

The following loads can be prescribed in a low-cycle fatigue analysis using the direct cyclic approach:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

During the analysis, each load must have an amplitude definition that is cyclic over the step where the start value must be equal to the end value (see *Amplitude Curves*). If the analysis consists of several steps, the usual rules apply (see *About Loads*). At each new step, the loading can either be modified or completely defined. All loads defined in previous steps remain unchanged unless they are redefined.

Predefined Fields

The following predefined fields can be specified in a low-cycle fatigue analysis using the direct cyclic approach, as described in *Predefined Fields*:

- Temperature is not a degree of freedom in a low-cycle fatigue analysis using the direct cyclic approach, but nodal temperatures can be specified as a predefined field. The temperature values specified must be cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see *Initial Conditions*). Alternatively, you can ramp the temperatures back to their initial condition values, as described in *Predefined Fields*. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. The field variable values specified must be cyclic over the step.

Material Options

Most ductile material models that describe mechanical behavior are available for use in a low-cycle fatigue analysis. The inelastic definition in a material point must be used in conjunction with the linear elastic material model (*Linear Elastic Behavior*), the porous elastic material model (*Elastic Behavior of Porous Materials*), or the hypoelastic material model (*Hypoelastic Behavior*).

The following material properties are not active during a low-cycle fatigue analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, piezoelectric properties, and pore fluid flow properties.

Rate-dependent yield (*Rate-Dependent Yield*), rate-dependent creep (*Rate-Dependent Plasticity: Creep and Swelling*), and two-layer viscoplasticity (*Two-Layer Viscoplasticity*) can also be used during a low-cycle fatigue analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a low-cycle fatigue analysis (see *Choosing the Appropriate Element for an Analysis Type*). This includes cohesive elements with finite thickness (*Modeling of an Adhesive Layer of Finite Thickness*). However, when modeling fatigue crack growth based on the principles of linear elastic fracture mechanics with the extended finite element method, only first-order continuum stress/displacement elements and second-order stress/displacement tetrahedron elements can be associated with an enriched feature (see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*).

Output

Different types of output are available for postprocessing and for monitoring a low-cycle fatigue analysis using the direct cyclic approach.

Message File Information

As in a direct cyclic analysis, low-cycle fatigue analysis using the direct cyclic approach in Abaqus/Standard prints the residual force, time average force, and a flag to indicate if equilibrium was satisfied in the message (`.msg`) file at different time increments for each iteration in each loading cycle. You can control the frequency in increments at which information is printed to the message file, and you can suppress the output; the default is to print output every 10 increments (see *The Abaqus/Standard Message File* for more information).

Abaqus/Standard also prints the number of Fourier terms used, the maximum residual coefficient, the maximum correction to displacement coefficients, and the maximum displacement coefficient in the Fourier series in the message file at the end of each iteration in each cycle. An example of the output is shown below:

```
                        CYCLE  5 STARTS

                     ITERATION    26 STARTS
INC     TIME         STEP        LARG. RESI.    TIME AVG.    FORCE
        INC          TIME        FORCE          FORCE        EQUV.
10      0.250        2.50        1.008E+01      50.9          N
20      0.250        5.00        1.622E+01      76.8          N
30      0.250        7.50        4.622E-02      99.8          Y

                     ITERATION    26 SUMMARY
NUMBER OF FOURIER TERMS USED 40, TOTAL NUMBER OF INCREMENTS  120
CYCLE/STEP TIME   30.0,    TOTAL TIME COMPLETED       31.0
AVERAGE FORCE     21.2     TIME AVG. FORCE     25.7
```

```
MAX. COEFFICIENT OF DISP.                    0.142  AT NODE 24 DOF 2
MAX. COEFF. OF RESI. FORCE ON CONST. TERM    31.7  AT NODE 44 DOF 1
MAX. COEFF. OF RESI. FORCE ON PERI. TERMS    0.82  AT NODE  6 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON CONST. TERM 0.002 AT NODE 50 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON PERI. TERMS 0.015 AT NODE 50 DOF 3
```

Results Output

Element and nodal output are written only when the stabilized cycle is reached. If a stabilized cycle has not been reached at the end of a cycle, output is written for the last iteration of the cycle. All standard output variables in Abaqus/Standard (*Abaqus/Standard Output Variable Identifiers*) are available. In addition, the following variables are available for progressive damage in bulk ductile material based on the continuum damage mechanics approach:

STATUS Status of element (the status of an element is 1.0 if the element is active, 0.0 if the element is not).

SDEG Scalar stiffness degradation, D.

CYCLEINI Number of cycles to initialize the damage at the material point.

The following variables are available for discrete crack propagation along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method:

STATUSXFEM Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked, 0.0 if the element is not. If the element is partially cracked, the value lies between 1.0 and 0.0.)

CYCLEINIXFEM Number of cycles to initialize the crack at the enriched element.

ENRRTXFEM All components of strain energy release rate range.

Recovering Additional Results for a Stabilized Cycle

You may want to recover additional results for a stabilized cycle. You can extract these results from the restart data (see *Recovering Additional Results Output from Restart Data in Abaqus/Standard*).

Input File Usage: **POST OUTPUT*, CYCLE=*n*

Abaqus/CAE Usage: Recovering additional results for a stabilized cycle is not supported in Abaqus/CAE.

Specifying Output at Exact Times

Output at exact times is not supported for low-cycle fatigue analysis. If output at exact times is requested, Abaqus will issue a warning message and change the output to an output at approximate times.

Limitations

A low-cycle fatigue analysis using the direct cyclic approach is subject to the following limitations:

- Contact conditions cannot change during a given cycle when direct cyclic analysis is used iteratively to obtain a stabilized solution.

- The analysis may not perform well when there is compressive load on the crack surface during a loading cycle because the global stiffness is formed only one time at the beginning of each given loading cycle.
- Geometric nonlinearity can be included only in any general step prior to a direct cyclic step; however, only small displacements and strains will be considered during the cyclic step.

Input File Template

The following is an example of modeling progressive damage and failure in the bulk material based on the continuum damage mechanics approach and progressive delamination growth at the interface:

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*MATERIAL
Options to define material properties
*DAMAGE INITIATION, CRITERION=HYSTERESIS ENERGY
Data lines to define material constants for bulk ductile material damage initiation
*DAMAGE EVOLUTION, TYPE=HYSTERESIS ENERGY
Data lines to define material constants for bulk ductile material damage evolution
**
*SURFACE, NAME=secondary
Data lines to define secondary surface at delamination interface
*SURFACE, NAME=main
Data lines to define main surface at delamination interface
*CONTACT PAIR
secondary, main
**
*STEP (,INC=)
Set  INC equal to the maximum number of increments in a single loading cycle
*DIRECT CYCLIC, FATIGUE
Data line to define time increment, cycle time, initial number of Fourier terms,
maximum number of Fourier terms, increment in number of Fourier terms,
and maximum number of iterations
Data line to define minimum increment in number of cycles,
maximum increment in number of cycles, total number of cycles,
and damage extrapolation tolerance
*DEBOND, SECONDARY=secondary, MAIN=main
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in Paris law and  fracture criterion
**
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
**
*END STEP
```

The following is an example of modeling discrete crack growth in the bulk material based on the principles of linear elastic fracture mechanics with the extended finite element method and progressive delamination growth at the interface:

```
*HEADING
…
*ENRICHMENT, TYPE=PROPAGATION CRACK, INTERACTION=INTERACTION,
ELSET=ENRICHED
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*MATERIAL
Options to define material properties
*SURFACE, INTERACTION=INTERACTION
*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in the Paris law and fracture criterion in the bulk
material for enriched elements
**
*SURFACE, NAME=secondary
Data lines to define secondary surface at delamination interface
*SURFACE, NAME=main
Data lines to define main surface at delamination interface
*CONTACT PAIR
secondary, main
**
*STEP (,INC=)
Set INC equal to the maximum number of increments in a single loading cycle
*DIRECT CYCLIC, FATIGUE
Data line to define time increment, cycle time, initial number of Fourier terms,
maximum number of Fourier terms, increment in number of Fourier terms,
and maximum number of iterations
Data line to define minimum increment in number of cycles,
maximum increment in number of cycles, total number of cycles,
and damage extrapolation tolerance
*DEBOND, SECONDARY=secondary, MAIN=main
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in the Paris law and fracture criterion at the interface
**
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
**
*END STEP
```

References

- Coffin, L., “A Study of the Effects of Cyclic Thermal Stresses on a Ductile Metal,” *Transactions of the American Society of Mechanical Engineering*, vol. 76, pp. 931–951, 1954.

- Manson, S., "Behavior of Materials under Condition of Thermal Stress," *Heat Transfer Symposium, University of Michigan Engineering Research Institute, Ann Arbor, MI*, pp. 9–75, 1953.
- Paris, P., M. Gomaz, and W. Anderson, "A Rational Analytic Theory of Fatigue," *The Trend in Engineering*, vol. 15, 1961.

Dynamic Stress/Displacement Analysis

In this section:

- *About Dynamic Analysis Procedures*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Explicit Dynamic Analysis*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Complex Eigenvalue Extraction*
- *Transient Modal Dynamic Analysis*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*
- *Response Spectrum Analysis*
- *Random Response Analysis*

About Dynamic Analysis Procedures

Overview

Abaqus offers several methods for performing dynamic analysis of problems in which inertia effects are considered.

Direct integration of the system must be used when nonlinear dynamic response is being studied. Implicit direct integration is provided in Abaqus/Standard; explicit direct integration is provided in Abaqus/Explicit. Modal methods are usually chosen for linear analyses because in direct-integration dynamics the global equations of motion of the system must be integrated through time, which makes direct-integration methods significantly more expensive than modal methods. Subspace-based methods are provided in Abaqus/Standard and offer cost-effective approaches to the analysis of systems that are mildly nonlinear.

In Abaqus/Standard dynamic studies of linear problems are generally performed by using the eigenmodes of the system as a basis for calculating the response. In such cases the necessary modes and frequencies are calculated first in a frequency extraction step. The mode-based procedures are generally simple to use; and the dynamic response analysis itself is usually not expensive computationally, although the eigenmode extraction can become computationally intensive if many modes are required for a large model. The eigenvalues can be extracted in a prestressed system with the "stress stiffening" effect included (the initial stress matrix is included if the base state step definition included nonlinear geometric effects), which might be necessary in the dynamic study of preloaded systems. It is not possible to prescribe nonzero displacements and rotations directly in mode-based procedures. The method for prescribing motion in mode-based procedures is explained in *Base motions in modal-based procedures*.

Density must be defined for all materials used in any dynamic analysis, and damping (both viscous and structural) can be specified either at the material or step level, as described below in *Damping in Dynamic Analysis*.

References:

Implicit Versus Explicit Dynamics

The direct-integration dynamic procedure provided in Abaqus/Standard offers a choice of implicit operators for integration of the equations of motion, while Abaqus/Explicit uses the central-difference operator. In an implicit dynamic analysis, the integration operator matrix must be inverted and a set of nonlinear equilibrium equations must be solved at each time increment. In an explicit dynamic analysis, displacements and velocities are calculated in terms of quantities that are known at the beginning of an increment; therefore, the global mass and stiffness matrices need not be formed and inverted, which means that each increment is relatively inexpensive compared to the increments in an implicit integration scheme. The size of the time increment in an explicit dynamic analysis is limited, however, because the central-difference operator is only conditionally stable; whereas the implicit operator options available in Abaqus/Standard are unconditionally stable and, thus, there is no such limit on the size of the time increment that can be used for most analyses in Abaqus/Standard (accuracy governs the time increment in Abaqus/Standard).

The stability limit for the central-difference method (the largest time increment that can be taken without the method generating large, rapidly growing errors) is closely related to the time required for a stress wave to cross the smallest element dimension in the model; thus, the time increment in an explicit dynamic analysis can be very short if the mesh contains small elements or if the stress wave speed in the material is very high. The method is, therefore, computationally attractive for problems in which the total dynamic response time that must be modeled is only a few orders of magnitude longer than this stability limit; for example, wave propagation studies or some "event and response" applications. Many of the advantages of the explicit procedure also apply to slower (quasi-static) processes for cases in which it is appropriate to use mass scaling to reduce the wave speed (see *Mass Scaling*).

Abaqus/Explicit offers fewer element types than Abaqus/Standard. For example, only first-order, displacement method elements (4-node quadrilaterals, 8-node bricks, etc.) and modified second-order elements are used, and each degree of freedom in the model must have mass or rotary inertia associated with it. However, the method provided in Abaqus/Explicit has some important advantages:

1. The analysis cost rises only linearly with problem size, whereas the cost of solving the nonlinear equations associated with implicit integration rises more rapidly than linearly with problem size. Therefore, Abaqus/Explicit is attractive for very large problems.
2. The explicit integration method is often more efficient than the implicit integration method for solving extremely discontinuous short-term events or processes.
3. Problems involving stress wave propagation can be far more efficient computationally in Abaqus/Explicit than in Abaqus/Standard.

In choosing an approach to a nonlinear dynamic problem, you must consider the length of time for which the response is sought compared to the stability limit of the explicit method; the size of the problem; and the restriction of the explicit method to first-order, pure displacement method or modified second-order elements. In some cases the choice is obvious, but in many problems of practical interest the choice depends on details of the specific case. Experience is then the only useful guide.

Direct-Solution Versus Modal Superposition Procedures

Direct solution procedures must be used for dynamic analyses that involve a nonlinear response. Modal superposition procedures are a cost-effective option for performing linear or mildly nonlinear dynamic analyses.

Direct-Solution Dynamic Analysis Procedures

The following direct-solution dynamic analyses procedures are available in Abaqus:

Implicit dynamic analysis

Implicit direct-integration dynamic analysis (*Implicit Dynamic Analysis Using Direct Integration*) is used to study (strongly) nonlinear transient dynamic response in Abaqus/Standard.

Subspace-based explicit dynamic analysis

The subspace projection method in Abaqus/Standard uses direct, explicit integration of the dynamic equations of equilibrium written in terms of a vector space spanned by a number of eigenvectors (*Implicit Dynamic Analysis Using Direct Integration*). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. This method can be very effective for systems with mild nonlinearities that do not substantially change the mode shapes. It cannot be used in contact analyses.

Explicit dynamic analysis

Explicit direct-integration dynamic analysis (*Explicit Dynamic Analysis*) is available in Abaqus/Explicit.

Direct-solution steady-state harmonic response analysis

The steady-state harmonic response of a system can be calculated in Abaqus/Standard directly in terms of the physical degrees of freedom of the model (*Direct-Solution Steady-State Dynamic Analysis*). The solution is given as in-phase (real) and out-of-phase (imaginary) components of the solution variables (displacement, stress, etc.) as functions of frequency. The main advantage of this method is that frequency-dependent effects (such as frequency-dependent damping) can be modeled. The direct method is the most accurate but also the most expensive steady-state harmonic response procedure. The direct

method can also be used if nonsymmetric terms in the stiffness are important or if model parameters depend on frequency.

Modal Superposition Procedures

Abaqus includes a full range of modal superposition procedures. Modal superposition procedures can be run using a high-performance linear dynamics software architecture called SIM. The SIM architecture offers advantages over the traditional linear dynamics architecture for some large-scale analyses, as discussed below in *Using the SIM Architecture for Modal Superposition Dynamic Analyses*.

Prior to any modal superposition procedure, the natural frequencies of a system must be extracted using the eigenvalue analysis procedure (*Natural Frequency Extraction*). Frequency extraction can be performed using the SIM architecture.

The following modal superposition procedures are available in Abaqus:

Mode-based steady-state harmonic response analysis

A steady-state dynamic analysis based on the natural modes of the system can be used to calculate a system's linearized response to harmonic excitation (*Mode-Based Steady-State Dynamic Analysis*). This mode-based method is typically less expensive than the direct method. The solution is given as in-phase (real) and out-of-phase (imaginary) components of the solution variables (displacement, stress, etc.) as functions of frequency. Mode-based steady-state harmonic analysis can be performed using the SIM architecture.

Subspace-based steady-state harmonic response analysis

In this type of Abaqus/Standard analysis, the steady-state dynamic equations are written in terms of a vector space spanned by a number of eigenvectors (*Subspace-Based Steady-State Dynamic Analysis*). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. The method is attractive because it allows frequency-dependent effects to be modeled and is much cheaper than the direct analysis method (*Direct-Solution Steady-State Dynamic Analysis*). Subspace-based steady-state harmonic response analysis can be used if the stiffness is nonsymmetric and can be performed using the SIM architecture.

Mode-based transient response analysis

The modal dynamic procedure (*Transient Modal Dynamic Analysis*) provides transient response for linear problems using modal superposition. Mode-based transient analysis can be performed using the SIM architecture.

Response spectrum analysis

A linear response spectrum analysis (*Response Spectrum Analysis*) is often used to obtain an approximate upper bound of the peak significant response of a system to a user-supplied input spectrum (such as earthquake data) as a function of frequency. The method has a very low computational cost and provides useful information about the spectral behavior of a system. Response spectrum analysis can be performed using the SIM architecture.

Random response analysis

The linearized response of a model to random excitation can be calculated based on the natural modes of the system (*Random Response Analysis*). This procedure is used when the structure is excited continuously and the loading can be expressed statistically in terms of a “Power Spectral Density” (PSD) function. The

response is calculated in terms of statistical quantities such as the mean value and the standard deviation of nodal and element variables. Random response analysis can be performed using the SIM architecture.

Complex eigenvalue extraction

The complex eigenvalue extraction procedure performs eigenvalue extraction to calculate the complex eigenvalues and the corresponding complex mode shapes of a system (*Complex Eigenvalue Extraction*). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. The complex eigenvalue extraction can be performed using the SIM architecture.

Using the SIM Architecture for Modal Superposition Dynamic Analyses

SIM is a high-performance software architecture available in Abaqus that can be used to perform modal superposition dynamic analyses. The SIM architecture is much more efficient than the traditional architecture for large-scale linear dynamic analyses (both model size and number of modes) with minimal output requests.

SIM-based analyses can be used to efficiently handle nondiagonal damping generated from element or material contributions, as discussed below in *Damping in a Mode-Based Steady-State and Transient Linear Dynamic Analysis Using the SIM Architecture*. Therefore, SIM-based procedures are an efficient alternative to subspace-based linear dynamic procedures for models with element damping or frequency-independent materials.

Activating the SIM Architecture

To use the SIM architecture for a modal superposition dynamic analysis, activate SIM for the initial frequency extraction procedure. SIM-based frequency extraction procedures write the mode shapes and other modal system information to a special linear dynamics data (`.sim`) file. By default, this data file is written to the scratch directory and deleted upon job completion; however, if restart is requested, the file is saved in the user directory. All subsequent mode-based steady-state or transient dynamic steps in an analysis automatically use this linear dynamics data file (and by extension the SIM architecture). If you restart an analysis that uses the SIM architecture, you must include the linear dynamics data file.

For more information about frequency extraction procedures, see *Natural Frequency Extraction*.

Input File Usage: **FREQUENCY*, SIM

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Use SIM-based linear dynamics procedures**

Example

The SIM architecture will be used for the entire linear dynamic analysis in the following input file template:

```
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS, SIM
Data line to control eigenvalue extraction
*COMPOSITE MODAL DAMPING
Data lines to define fraction of critical damping
*END STEP
**
*STEP
*MODAL DYNAMIC
Data line to control time incrementation
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING, VISCOUS=COMPOSITE
```

```
Data lines to define composite modal damping
*END STEP
**
*STEP
*STEADY STATE DYNAMICS
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*END STEP
```

Output in a SIM-Based Analysis

Output is a fundamental factor in the performance of a linear dynamic analysis. Since it is difficult to predict the desired output quantities for a linear dynamic analysis, preselected output requests are ignored in SIM-based modal superposition procedures (except complex eigenvalue extraction). You must always specify output requests to the output database (`.odb`) file; otherwise, the analysis will not be performed.

There are several restrictions on available output requests that apply specifically to SIM-based analyses:

- You cannot request output to the results (`.fil`) file.
- Element variables cannot be output to the printed data (`.dat`) file except for random response analysis.

Limitations of the SIM Architecture

The cyclic symmetry modeling feature cannot be used in SIM-based analyses.

Reaction Force Calculations in Mode-Based Dynamic Analyses

In modal procedures that do not use SIM-based analysis, the reaction force calculation is based on the modal reaction forces extracted in the frequency extraction procedure. This approach does not take into account nondiagonal mass matrix and damping matrix contributions to the reaction force (as in the case of structural elements or substructures). Therefore, it might give rise to incorrect reaction force results. It is recommended that you use the steady-state dynamic or transient dynamic procedures based on the SIM architecture.

Nonphysical Material Properties in Dynamic Analyses

Abaqus relies on user-supplied model data and assumes that the material's physical properties reflect experimental results. Examples of meaningful material properties are a positive mass density per volume, a positive Young's modulus, and a positive value for any available damping coefficients. However, in special cases you might want to “adjust” a value of density, mass, stiffness, or damping in a region or a part of the model to bring the overall mass, stiffness, or damping to the expected required levels. Certain material options in Abaqus allow you to introduce nonphysical material properties to achieve this adjustment.

For example, to adjust the mass of the model, you can define a nonstructural mass with a negative mass value, use mass elements with a negative mass over a region of nodes, or introduce additional elements with negative density. Similarly, to adjust damping levels, you can use negative damping coefficients or introduce dashpot elements with a negative dashpot constant to reduce the overall damping levels. Springs with negative stiffness can be defined to adjust the model stiffness.

If you specify nonphysical but allowed material properties, Abaqus issues a warning message. However, if you specify nonphysical material properties that are not allowed, Abaqus issues an error message. When introducing nonphysical material properties, you must be aware that the overall behavior should be “physical”; for example, the mass values at all nodes must be positive in an eigenvalue extraction procedure.

There are consequences of using nonphysical material properties that are easy to check and interpret, and there are others beyond the control of Abaqus. Therefore, you should fully understand the stated problem and the consequences of using nonphysical material properties before you specify the properties. This is particularly important in Abaqus/Explicit analyses, where the size of the time increment depends on material properties. For example, distributed mass-dependent loads are calculated based on the overall mass density (positive and negative) provided.

Damping in Dynamic Analysis

Every nonconservative system exhibits some energy loss that is attributed to material nonlinearity, internal material friction, or to external (mostly joint) frictional behavior. Conventional engineering materials like steel and high strength aluminum alloys provide small amounts of internal material damping, not enough to prevent large amplification at or near resonant frequencies. Damping properties increase in modern composite fiber-reinforced materials, where the energy loss occurs through plastic or viscoelastic phenomena as well as from friction at the interfaces between the matrix and reinforcement. Still larger material damping is exhibited by thermoplastics. Mechanical dampers can be added to models to introduce damping forces to the system. In general, it is difficult to quantify the source of a system's damping. It usually comes from several sources simultaneously; for example, from energy loss during hysteretic loading, viscoelastic material properties, and external joint friction.

Users that work with a specific system know the source of the energy loss from experience. A variety of methods are available in Abaqus to specify damping that accurately models the energy loss in a dynamic system.

Sources of Damping

Abaqus has four categories of damping sources: material and element damping, global damping, modal damping, and damping associated with time integration. If necessary, you can include multiple damping sources and combine different damping sources in a model.

Material and Element Damping

Damping can be specified as part of a material definition that is assigned to a model (see *Material Damping*). In addition, Abaqus has elements such as dashpots, springs with their complex stiffness matrix, and connectors that serve as dampers, all with viscous and structural damping factors. Viscous damping can be included in mass, beam, pipe, and shell elements with general section properties; and it can also be used in substructure elements (see *Generating Substructures*). In direct steady-state dynamic analysis, you can define the viscous and structural damping due to the interaction between the contacting surfaces by using user subroutine *UINTER* (see *UINTER*). Contact damping is not applicable for linear perturbation procedures.

In acoustic elements, velocity proportional viscous damping is implemented using the volumetric drag parameter (see *Acoustic Medium*). Acoustic infinite elements and impedance conditions also add damping to a model.

Global Damping

In situations where material or element damping is not appropriate or sufficient, you can apply abstract damping factors to an entire model. Abaqus allows you to specify global damping factors for both viscous (Rayleigh damping) and structural damping (imaginary stiffness matrix).

Modal Damping

Modal damping applies only to mode-based linear dynamic analyses. This technique allows you to apply damping directly to the modes of the system. By definition, modal damping contributes only diagonal entries to the modal system of equations and can be defined several different ways.

Damping Associated with Time Integration

Marching through a simulation with a finite time increment size causes some damping. This type of damping applies only to analyses using direct time integration. See *Implicit Dynamic Analysis Using Direct Integration* for further discussion of this source of damping.

Damping in a Linear Dynamic Analysis

Damping can be applied to a linear dynamic system in two forms:

- velocity proportional viscous damping; and
- displacement proportional structural damping, which is for use in frequency domain dynamics. The exception is SIM-based transient modal dynamic analysis, where the structural damping is converted to the equivalent diagonal viscous damping (see *Modal dynamic analysis*).

An additional type of damping known as composite damping serves as a means to calculate a model average critical damping with the material density as the weight factor and is intended for use in mode-based dynamics (excluding subspace projection steady-state analysis). For more information, see *Damping options for modal dynamics*.

The types of damping available for linear dynamic analyses depend on the procedure type and the architecture (traditional or SIM) used to perform the analysis, as outlined in *Table 1* and *Table 2*. For completeness, *Table 1* also includes the damping options for a direct steady-state dynamic analysis. In addition to directly specified modal damping, global damping can be used in all linear dynamic procedures. Material and element damping can be used in subspace-based and SIM-based linear dynamic procedures.

Table 1: Damping sources for traditional architecture.

Traditional Architecture	Damping Source		
	Modal	Global	Material and Element
Mode-based steady-state dynamics	√	√	
Subspace-based steady-state dynamics		√	√
Transient modal dynamics	√	√	
Random response analysis	√	√	
Complex frequency		√	√
Response spectrum	√	√	
Direct steady-state dynamics		√	√

Table 2: Damping sources for SIM architecture.

SIM Architecture	Damping Source		
	Modal	Global	Material and Element
Mode-based steady-state dynamics	√	√	√
Subspace-based steady-state dynamics	√	√	√
Transient modal dynamics	√	√	√
Random response analysis	√	√	
Complex frequency	√	√	√
Response spectrum	√	√	

In a subspace-based or SIM-based linear dynamic analysis, material and element damping operators must first be projected onto the basis of mode shapes. This projection results in a full modal damping matrix for both viscous and structural damping; therefore, a modal steady-state response analysis requires the solution of a system of linear equations at each frequency point. The size of this system is equal to the number of modes used in the response calculation. In a mode-based transient analysis, the projected damping operator is treated explicitly in time by including it on the right-hand side of the system of equations.

Frequency-dependent damping is supported only for the subspace-based and direct-integration steady-state dynamic procedures.

Material and element damping is not supported for the response spectrum or the random response procedures. In these procedures, only modal and global damping are allowed, and material or element damping is ignored.

Damping in a Mode-Based Steady-State and Transient Linear Dynamic Analysis Using the SIM Architecture

SIM-based linear dynamic analyses may include material and element damping contributions that introduce both diagonal and nondiagonal terms in the modal system of equations. The projection of material and element damping operators onto the basis of mode shapes is performed during the natural frequency extraction procedure, which enables a high-performance projection operation to be performed when used with the AMS eigensolver. If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

When the structural and viscous damping operators are projected onto the mode shapes, the full modal damping matrix is stored in the linear dynamics data (`.sim`) file. The full modal damping matrix is combined with any diagonal contributions from global damping or traditional modal damping. The combined damping operator matrix is included in subsequent mode-based transient or steady-state dynamics steps. If there are nondiagonal (that is, projected) damping contributions and a large number of modes are included, performance of the linear dynamics calculations will be impacted since a direct solve must be performed at each frequency point.

Acoustic damping due to impedance conditions is projected onto the subspace of acoustic eigenvectors. These contributions are taken into account in a subspace-based steady-state dynamics analysis that uses the SIM architecture.

The default behavior for a SIM-based frequency extraction step is to project any element and material damping onto the mode shapes. You can turn off this damping projection if it is not desired; however, in this case only diagonal damping is available for subsequent modal superposition steps. If the projected damping matrices are not desired in a particular mode-based linear dynamic step for performance reasons, they can be deactivated in that step using the damping control techniques discussed above in *Damping in Dynamic Analysis*.

Input File Usage: Use the following option to project material and element damping operators in a SIM-based analysis:

**FREQUENCY*, SIM, DAMPING PROJECTION=ON (default)

Use the following option to turn off damping projection in a SIM-based analysis:

**FREQUENCY*, SIM, DAMPING PROJECTION=OFF

Abaqus/CAE Usage: To control the projection of element and material damping in a SIM-based frequency extraction step that uses the Lanczos eigensolver:

Step module: **Step->Create**: **Frequency**: **Eigensolver: Lanczos**, **Use SIM-based linear dynamics procedures**, toggle **Project damping operators**

To control the projection of element and material damping in a frequency extraction step that uses the AMS eigensolver:

Step module: **Step->Create**: **Frequency**: **Eigensolver: AMS**, toggle **Project damping operators**

Defining Viscous Damping

Abaqus allows you to choose a particular source of viscous damping, to add several sources, or to exclude viscous damping effects.

Defining Material/Element Viscous Damping

You can choose to model the viscous damping matrix, $\boldsymbol{D}_{\boldsymbol{viscous}}^{\boldsymbol{el}}$, by using material damping properties and/or damping elements (such as dashpot or mass elements). The viscous, mass, and/or stiffness proportional damping matrix will include the material Rayleigh damping factors, $\boldsymbol{\alpha}_{\boldsymbol{R}}^{\boldsymbol{mat}}$ and $\boldsymbol{\beta}_{\boldsymbol{R}}^{\boldsymbol{mat}}$, as well as the element-oriented damping factor, $\boldsymbol{\alpha}_{\boldsymbol{R}}^{\boldsymbol{el}}$ (for example, for mass elements). The material/element-based viscous damping matrix can be written as

$$\begin{aligned}\boldsymbol{D}_{\boldsymbol{viscous}}^{\boldsymbol{el}} &= \sum_{\boldsymbol{el}=1}^{\boldsymbol{Num\ elems}} \int_{\boldsymbol{V}} \boldsymbol{\alpha}_{\boldsymbol{R}}^{\boldsymbol{mat}} \boldsymbol{N}^{\boldsymbol{T}} \boldsymbol{N} \boldsymbol{\rho} \boldsymbol{dv} + \sum_{\boldsymbol{el}=1}^{\boldsymbol{Num\ elems}} \boldsymbol{\alpha}_{\boldsymbol{R}}^{\boldsymbol{el}} \boldsymbol{m}^{\boldsymbol{el}} \\ &+ \sum_{\boldsymbol{el}=1}^{\boldsymbol{Num\ elems}} \int_{\boldsymbol{V}} \boldsymbol{\beta}_{\boldsymbol{R}}^{\boldsymbol{mat}} \boldsymbol{B}^{\boldsymbol{T}} \boldsymbol{D} \boldsymbol{B} \boldsymbol{dv} + \sum_{\boldsymbol{el}=1}^{\boldsymbol{Num\ elems}} \boldsymbol{d}_{\boldsymbol{viscous}}^{\boldsymbol{el}},\end{aligned}$$

where $\boldsymbol{d}_{\boldsymbol{viscous}}^{\boldsymbol{el}}$ represents the viscous damping matrix for elements such as dashpots. In mode-based procedures projection of $\boldsymbol{D}_{\boldsymbol{viscous}}^{\boldsymbol{el}}$ into the eigenmodes results in a nondiagonal matrix.

Input File Usage: Use the following option to specify material viscous damping for elements with mechanical degrees of freedom:

**DAMPING*, ALPHA=$\boldsymbol{\alpha}_{\boldsymbol{R}}^{\boldsymbol{mat}}$, BETA=$\boldsymbol{\beta}_{\boldsymbol{R}}^{\boldsymbol{mat}}$

Use the following option to specify material viscous damping for acoustic elements:

**ACOUSTIC MEDIUM*, VOLUMETRIC DRAG

Abaqus/CAE Usage: Property module: material editor: **Mechanical**->**Damping**: **Alpha**: α_R^{mat} or **Beta**: β_R^{mat}

Property module: material editor: **Other**->**Acoustic Medium**: **Volumetric Drag**

Defining Global Viscous Damping

You can supply global mass and stiffness proportional viscous damping factors, α_{global} and β_{global}, respectively, to create the global damping matrix using the global model mass and stiffness matrices, M and K, respectively:

$$D^g_{viscous} = \alpha_{global} M + \beta_{global} K.$$

These parameters can be specified for the entire model (default), for the mechanical degree of freedom field (displacements and rotations) only, or for the acoustic field only.

Input File Usage: Use the following option to specify global viscous damping:

**GLOBAL DAMPING*, ALPHA=α_{global}, BETA=β_{global}

Abaqus/CAE Usage: Global viscous damping is not supported in Abaqus/CAE.

Defining Viscous Modal Damping

Rayleigh damping introduces a damping matrix, $[C]$, defined as

$$[C] = \alpha [M] + \beta [K],$$

where $[M]$ is the mass matrix of the model, $[K]$ is the stiffness matrix of the model, and α and β are factors that you define.

In Abaqus/Standard you can define α and β independently for each mode, so that the above equation becomes

$$c_M = \alpha_M m_M + \beta_M k_M \quad \text{(no sum on M)},$$

where the subscript M refers to the mode number and c_M, m_M, and k_M are the damping, mass, and stiffness terms associated with the Mth mode.

Input File Usage: Use the following option to define Rayleigh damping by specifying mode numbers:

**MODAL DAMPING*, VISCOUS=RAYLEIGH, DEFINITION=MODE NUMBERS

Use the following option to define Rayleigh damping by specifying a frequency range:

**MODAL DAMPING*, VISCOUS=RAYLEIGH, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define Rayleigh damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Rayleigh**: **Use Rayleigh damping data**

Use the following input to define Rayleigh damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Rayleigh**: **Use Rayleigh damping data**

Defining Viscous Modal Damping as a Fraction of the Critical Damping

You can also specify the damping in each eigenmode in the model or for the specified frequency as a fraction of the critical damping. Critical damping is defined as

$$c_{cr} = 2\sqrt{mk},$$

where m is the mass of the system and k is the stiffness of the system. Typical values of the fraction of critical damping, ξ_i, are from 1% to 10% of critical damping, c_{cr}; but Abaqus/Standard accepts any positive value. The critical damping factors can be changed from step to step.

Input File Usage: Use the following option to define the fraction of critical damping by specifying mode numbers:

**MODAL DAMPING*, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define the fraction of critical damping by specifying a frequency range:

**MODAL DAMPING*, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define the fraction of critical damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Direct modal**: **Use direct damping data**

Use the following input to define the fraction of critical damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Direct modal**: **Use direct damping data**

Viscous Modal Damping for Uncoupled Structural-Acoustic Frequency Extractions

For uncoupled structural-acoustic frequency extractions performed using the AMS eigensolver, you can apply different damping to the structural and acoustic modes. This technique can be used only when damping is specified for a range of frequencies.

Input File Usage: Use the following option to apply the specified damping to only the structural modes:

**MODAL DAMPING*, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=MECHANICAL

Use the following option to apply the specified damping to only the acoustic modes:

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=ACOUSTIC

Use the following option to apply the specified damping to both structural and acoustic modes (default):

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=ALL

Abaqus/CAE Usage: The ability to specify different damping for structural and acoustic modes is not supported in Abaqus/CAE.

Controlling the Sources of Viscous Damping

The material/element and global viscous damping sources can be controlled at the step level; controls are not available for modal damping. If both the material/element and global viscous damping matrices are supplied, both will be used as a combined damping matrix unless you request that only the element or global damping factor be used. The combined material/element and global viscous damping is

$$D_{viscous} = D^{el}_{viscous} + D^{g}_{viscous}.$$

Input File Usage: Use the following option to activate only the material/element viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=ELEMENT

Use the following option to activate only the global viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=FACTOR

Use the following option to activate the combined material/element and global viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=COMBINED

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Excluding Viscous Damping Effects

You can choose to exclude the effects of viscous damping altogether at the step level.

Input File Usage: Use the following option to exclude the viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Defining Structural Damping

Abaqus allows you to choose a particular source of structural damping, to add several sources, or to exclude structural damping effects.

Defining Material/Element Structural Damping

The material/element structural damping matrix (that represents the imaginary stiffness and is proportional to forces or displacements) is defined as

$$K_s^m = \sum_{el=1}^{Num\ elems} \int_V \mathbf{s} B^T D B dv + \sum_{el=1}^{Num\ elems} \mathbf{s}^{el} k^{el},$$

where $\mathbf{s}$ represents the material structural damping, $\mathbf{s}^{el}$ represents the structural damping coefficient for elements such as springs with complex stiffnesses and connectors, and k^{el} is the real element stiffness matrix. In mode-based procedures the projection of K_s^m onto the mode shapes results in a full modal damping matrix. When using SIM-based modal procedures, the projected material and element damping matrix may be combined with global and modal damping (see *Defining and Using Both Global and Modal Diagonal Damping* below). Material/element structural damping is not available for acoustic elements.

Input File Usage: Use the following option to specify material structural damping:

**DAMPING*, STRUCTURAL=$\mathbf{s}$

Abaqus/CAE Usage: Property module: material editor: **Mechanical**->**Damping**: **Structural**: $\mathbf{s}$

Defining Global Structural Damping

You can define the global structural damping factor, $\mathbf{s}_{global}$, to get

$$K_s^g = \mathbf{s}_{global} K.$$

Global structural damping can be specified for the entire model (default), for the mechanical degree of freedom field (displacements and rotations) only, or for the acoustic field only.

Input File Usage: Use the following option to specify global structural damping:

**GLOBAL DAMPING*, STRUCTURAL=$\mathbf{s}_{global}$

Abaqus/CAE Usage: Global structural damping is not supported in Abaqus/CAE.

Defining Structural Modal Damping

Structural damping assumes that the damping forces are proportional to the forces caused by stressing of the structure and are opposed to the velocity (see *Structural Damping* for more information). This form of damping can be used only when the displacement and velocity are exactly 90° out of phase, as in steady-state and random response analyses where the excitation is purely sinusoidal.

Structural damping can be defined as diagonal modal damping for mode-based steady-state dynamic and random response analyses.

Input File Usage: Use the following option to define structural damping by specifying mode numbers:

**MODAL DAMPING*, STRUCTURAL, DEFINITION=MODE NUMBERS

Use the following option to define structural damping by specifying a frequency range:

*MODAL DAMPING, STRUCTURAL,
DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define structural damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Structural**: **Use structural damping data**

Use the following input to define structural damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Structural**: **Use structural damping data**

Controlling the Sources of Structural Damping

The material/element and global structural damping sources can be controlled at the step level; controls are not available for modal damping. If both the material/element and global structural damping matrices are supplied, both will be combined unless you request that only the element or global damping factor be used. The combined structural damping matrix is

$$\boldsymbol{K}_s = \boldsymbol{K}_s^m + \boldsymbol{K}_s^g.$$

Input File Usage: Use the following option to activate only the material/element structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=ELEMENT

Use the following option to activate only the global structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=FACTOR

Use the following option to activate the combined material/element and global structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=COMBINED

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Excluding Structural Damping Effects

You can choose to exclude the effects of structural damping altogether at the step level.

Input File Usage: Use the following option to exclude structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Defining Both Viscous and Structural Damping

The imaginary contribution to the frequency domain dynamics equation, which represents the effect of damping, may include both viscous and structural damping and can be written as

$$D = \Omega D_{viscous} + K_s,$$

where Ω is the forcing frequency.

Defining Composite Modal Damping

Composite modal damping allows you to define a damping factor for each material or element in the model as a fraction of critical damping. These factors are then combined into a damping factor for each mode as weighted averages of the mass matrix associated with each material or element; when using the SIM architecture, you can also include the weighted averages of the stiffness matrix. Composite modal damping can be defined only by specifying mode numbers; it cannot be defined by specifying a frequency range.

Defining Composite Modal Damping for Analyses Using the Traditional Architecture

You specify composite modal damping in the material definition for analyses that use the traditional architecture. The damping per eigenmode is calculated as:

$$\xi_\alpha = \frac{1}{m_\alpha}\phi_\alpha^T \left(\sum_m \xi_m M_m \right) \phi_\alpha,$$

where ξ_α is the critical damping fraction used in mode α, ξ_m is the critical damping fraction defined for material m, M_m is the mass matrix associated with material m, ϕ_α is the eigenvector of mode α, and m_α is the generalized mass associated with mode α:

$$m_\alpha = \phi_\alpha^T M \phi_\alpha.$$

If you specify composite modal damping, Abaqus calculates the damping coefficients ξ_α in the eigenfrequency extraction step from the damping factors ξ_m that you defined for each material.

Input File Usage: Use both of the following options:

**DAMPING*, COMPOSITE=ξ_m
**MODAL DAMPING*, VISCOUS=COMPOSITE

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damping**: **Composite: ξ_m**
Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Composite modal**: **Use composite damping data**

Defining Composite Modal Damping for Analyses Using the SIM Architecture

You can specify composite modal damping for SIM-based analyses except when you use the AMS eigensolver and request selective recovery. Composite modal damping is specified for each element. You can also assign critical damping fractions to both mass and stiffness matrix input. The mass weighted damping per eigenmode is calculated as:

$$\xi_\alpha^M = \phi_\alpha^T M_\xi \phi_\alpha = \phi_\alpha^T \left(\sum_{elements} \xi_{element}^M m_{element} + \xi_{\substack{matrix \\ input}}^M M_{\substack{matrix \\ input}} \right) \phi_\alpha,$$

where ξ_α^M is the mass weighted critical damping fraction used in mode α, M_ξ is a damped portion of the mass matrix, ξ^M are fractions of critical damping for the element mass matrix and mass matrix input, and ϕ_α is the eigenvector of mode α.

The stiffness weighted damping per eigenmode is calculated as:

$$\xi_\alpha^K = \frac{1}{\omega_\alpha^2}\phi_\alpha^T K_\xi \phi_\alpha = \frac{1}{\omega_\alpha^2}\phi_\alpha^T \left(\sum_{elements} \xi_{element}^K k_{element} + \xi_{\substack{matrix \\ input}}^K K_{\substack{matrix \\ input}} \right) \phi_\alpha,$$

where ξ_α^K is the stiffness weighted critical damping fraction used in mode α, K_ξ is a damped portion of the stiffness matrix, ξ^K are fractions of critical damping for the element stiffness and matrix input stiffness, and ϕ_α is the eigenvector of mode α.

Input File Usage: Use both of the following options to specify composite modal damping:

**FREQUENCY*, EIGENSOLVER=LANCZOS, SIM
**COMPOSITE MODAL DAMPING*

Use the following option to specify the fraction of critical damping for all mass matrices included in matrix input:

**COMPOSITE MODAL DAMPING*, MASS MATRIX INPUT

Use the following option to specify the fraction of critical damping for all stiffness matrices included in matrix input:

**COMPOSITE MODAL DAMPING*, STIFFNESS MATRIX INPUT

Abaqus/CAE Usage: Defining composite modal damping for analyses using the SIM architecture is not supported in Abaqus/CAE.

Defining Global Damping in Acoustic Models

If your model contains only acoustic elements, Abaqus applies any specified global damping to all the acoustic fields by default. Similarly, if your model contains only stress/displacement elements, Abaqus applies any specified global damping to all the displacement and rotation fields by default.

If your model contains both acoustic elements and stress/displacement elements, the analysis type determines how global damping is applied. You can apply global damping to all of the displacement, rotation, and acoustic fields; to only the acoustic fields; or to only the displacement and rotation fields in the following procedures:

- Mode-based analyses using uncoupled modes and the default high-performance linear dynamics implementation during the frequency extraction
- Subspace-based steady-state dynamic analyses using coupled modes
- Direct steady-state analyses

You can apply global damping only to all of the displacement, rotation, and acoustic fields in the following procedures:

- Steady-state dynamic analyses using coupled modes
- Mode-based steady-state dynamic analyses using coupled acoustic-structural modes

Input File Usage: Use the following option to apply global damping to all of the displacement, rotation, and acoustic fields in a model:

**GLOBAL DAMPING*, FIELD=ALL (default)

Use the following option to apply global damping only to the acoustic fields:

**GLOBAL DAMPING*, FIELD=ACOUSTIC

Use the following option to apply global damping only to the displacement and rotation fields:

**GLOBAL DAMPING*, FIELD=MECHANICAL

Abaqus/CAE Usage: Global damping is not supported in Abaqus/CAE.

Defining Modal Damping in Acoustic Models

If your model contains only acoustic elements, Abaqus applies any specified modal damping to all the acoustic fields by default. Similarly, if your model contains only stress/displacement elements, Abaqus applies any specified modal damping to all the displacement and rotation fields by default.

If your model contains both acoustic elements and stress/displacement elements, you can apply modal damping to all of the displacement, rotation, and acoustic fields. However, you can apply modal damping to the displacement and rotation fields or the acoustic fields separately only when using uncoupled modes and the default high-performance linear dynamics implementation during the frequency extraction.

Input File Usage: Use the following option to apply modal damping to all of the displacement, rotation, and acoustic fields in a model:

**MODAL DAMPING*, FIELD=ALL (default)

Use the following option to apply modal damping only to the acoustic fields:

**MODAL DAMPING*, FIELD=ACOUSTIC

Use the following option to apply modal damping only to the displacement and rotation fields:

**MODAL DAMPING*, FIELD=MECHANICAL

Abaqus/CAE Usage: Modal damping is not supported in Abaqus/CAE.

Defining and Using Both Global and Modal Diagonal Damping

Mode-based procedures—such as steady-state dynamics, transient modal dynamic, response spectrum, and random response analyses—can also use a step-dependent, modal damping definition that is specified per eigenmode. When multiple modal damping definitions are used with different damping types, the damping is additive. If the same damping type is specified more than once, the last specification is used. If modal damping is used with global damping, both types of damping will contribute to the damping matrix.

Damping controls have no effect on modal damping. If damping controls are used to exclude certain global damping effects in a step, all modal damping effects are still included in the step. To exclude modal damping, the damping definition must be specifically removed from the step definition.

Controlling Damping of Low Frequency Modes

You can include or exclude damping of the low frequency eigenmodes in transient modal analyses. This control is useful for free structures and models with secondary base motions. For details, see *Transient Modal Dynamic Analysis*.

Acoustic Contribution Factors in Mode-Based and Subspace-Based Steady-State Dynamic Analyses

You can compute acoustic contribution factors for the linear, eigenmode-based, steady-state dynamic procedures. Computation of the acoustic contribution factors makes it possible to answer the following questions:

- Which noise source has the greatest contribution?
- Which point does the loudest noise come from?
- Which structural component does the loudest noise come from?
- Which natural frequency contributes the most to the noise?

By answering these questions, you can determine the major noise sources as well as make the necessary changes to your design to reduce the noise levels. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. It is available in steady-state mode-based and subspace-based dynamic analyses performed using the high-performance SIM architecture. To enable computation of the acoustic contribution factors, the preceding frequency extraction step has to use the uncoupled modes formulation as well as to activate the SIM architecture. Abaqus/Standard supports the computation of the following contribution factors:

- Acoustic modal contribution factors,
- Acoustic structural modal contribution factors,
- Acoustic load modal contribution factors,
- Acoustic load contribution factors,
- Panel contribution factors, and
- Grid contribution factors.

The acoustic node set defines a set of the response nodes in the acoustic domain. You can specify up to 20 response nodes in this set. You can also select the acoustic or structural eigenmodes that will be used to compute the modal contribution factors. You specify the lower and upper bounds of the frequency range to apply to the active eigenmodes (see *Selecting the Modes and Specifying Damping* and *Selecting the Modes on Which to Project*).

The computed contribution factors are saved in the SIM database file. You can retrieve the data as described in "Plug-in utility for visualizing Acoustic Contribution Factors" in the Dassault Systèmes Knowledge Base at *https://support.3ds.com/knowledge-base/*.

Alternatively, you can save the computed contribution factors in the output database (`.odb`) file. If you request history output, the computed acoustic contribution factors are saved as history curves in the output database. Field output is relevant only for the grid acoustic contribution factors. If you request field output for the grid acoustic contribution factors, their output into the SIM database file is blocked.

Input File Usage: Use the following option to request computation of the acoustic modal contribution factors:

**ACOUSTIC CONTRIBUTION*, NAME=*name*,
ACOUSTIC NODES=*acoustic_nset*
f_{min}, f_{max}

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Acoustic Modal Contribution Factors

Acoustic modal contribution factors show the contribution of each acoustic mode into the total acoustic pressure at the response points. You can also select the acoustic eigenmodes that will be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic modal contribution factors:

**ACOUSTIC CONTRIBUTION*, TYPE=MODAL ACOUSTIC

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Acoustic Structural Modal Contribution Factors

Acoustic structural modal contribution factors measure the contribution of each structural mode into the acoustic pressure caused by the structural components. You can also select the structural eigenmodes that will be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic structural modal contribution factors:

**ACOUSTIC CONTRIBUTION*, TYPE=MODAL STRUCTURAL

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Acoustic Load Modal Contribution Factors

Acoustic load modal contribution factors define the contribution of each acoustic mode due to the acoustic sources into the acoustic pressure. You can specify the acoustic eigenmodes that are going to be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic load modal contribution factors:

**ACOUSTIC CONTRIBUTION*, TYPE=MODAL LOAD

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Acoustic Load Contribution Factors

Acoustic load contribution factors define the contribution of the acoustic sources into the acoustic pressure.

Input File Usage: Use the following option to compute the acoustic load contribution factors:

**ACOUSTIC CONTRIBUTION*, TYPE=LOAD

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Panel Contribution Factors

Panel contribution factors measure the contribution of the user-defined structural surfaces into the acoustic pressure caused by structural sources.

Optionally, you can specify a set of nodes that defines a structural panel—a set of nodes on the acoustic-structural interface that is considered to be a single noise source. You can specify this set by giving the name of the node set containing the nodes on the acoustic-structural interface, or you can specify the set by giving the name of the surface containing the structural nodes on the acoustic-structural interface. If you use both methods simultaneously, the node set for computing panel acoustic contribution factors is imposed as the union of the node sets prescribed by each method. If this node set contains other structural or acoustic nodes that do not belong to the acoustic-structural interface, those nodes are filtered out and are not considered for panel contribution factor computations. If you do not specify a set of nodes on the acoustic-structural interface, all nodes on the interface are used to determine the panel contribution factors.

Input File Usage: Use the following option to compute the panel contribution factor:

**ACOUSTIC CONTRIBUTION*, TYPE=PANEL,
STRUCTURAL NODES=*structural_nset*, SURFACE=*structural_surface*

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying Grid Contribution Factors

Grid contribution factors measure the contribution of each node on the acoustic-structural interface into the acoustic pressure caused by structural sources.

Optionally, you can specify a set of nodes on the acoustic-structural interface. Each node in this set is considered to be an individual noise source. You can specify this set by giving the name of the node set containing the nodes on the acoustic-structural interface, or you can specify the set by giving the name of the surface containing the structural nodes on the acoustic-structural interface. If you use both methods simultaneously, the node set for computing panel acoustic contribution factors is imposed as the union of the node sets prescribed by each method. If this node set contains other structural or acoustic nodes that do not belong to the acoustic-structural interface, these nodes are filtered out and will not be considered for the grid contribution factor computations. If you do not specify a set of nodes on the acoustic-structural interface, all nodes on the interface are used to determine the grid contribution factors. Due to the large amount of data generated for grid contribution factors, the number of nodes in this node set is limited to 1,000,000 nodes. If this number is exceeded, the first 1,000,000 nodes are used.

Input File Usage: Use the following option to compute the grid contribution factor:

**ACOUSTIC CONTRIBUTION*, TYPE=GRID,
STRUCTURAL NODES=*structural_nset*, SURFACE=*structural_surface*

Input File Usage: Use the following option to compute the grid contribution factor and output it as field output to the output database (`.odb`) file:

**ACOUSTIC CONTRIBUTION*, TYPE=GRID,
STRUCTURAL NODES=*structural_nset*, SURFACE=*structural_surface*,
OUTPUT=FIELD

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Normal Velocity Output in Steady-State Dynamic Analysis

For metal structures in air, the forced response of the structure is largely uncoupled from the acoustic vibrations of the air in the frequency range of interest. On the other hand, the acoustic vibrations of the air are strongly driven by the structural motion. Therefore, it is common in engineering to estimate the noise radiated from a structure using only the surface normal velocity of the structure.

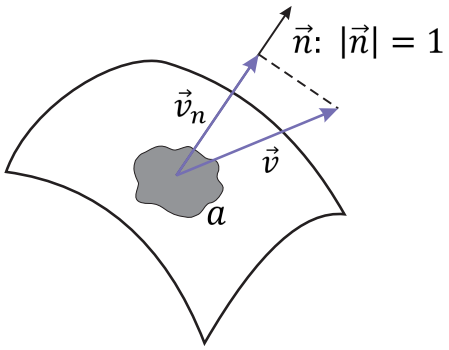
Abaqus/Standard can provide normal velocity field output at nodes of the element-based surface in a steady-state dynamic analysis (see *Figure 1*).

The following output variables are available:

VN The complex-valued surface normal velocity defined as the projection of the nodal complex velocity vector along the local (real-valued) surface normal vector at the node. VN is itself vector-valued, carrying the direction of the normal.

VNSQ The real-valued surface normal velocity squared, defined as the scalar (real) magnitude squared of the surface normal velocity vector.

AVNSQ The area-weighted surface normal velocity squared, or the acoustic power normalized by the acoustic impedance of the surrounding fluid; that is, a real-valued scalar equal to VNSQ multiplied by the local surface area adjacent to the node.



$$VN = \vec{v}_n = v_n\vec{n} \quad complex\ vector$$
$$VNSQ = |\vec{v}_n|^2 = v_n^* v_n \quad real\ scalar$$
$$AVNSQ = a|\vec{v}_n|^2 \quad real\ scalar$$

Figure 1: Normal velocity.

Input File Usage: Use the following option to specify the element-based surface for the normal velocity output:

**NODE OUTPUT*, SURFACE=*element_based_surface_name*
list of output variables

Abaqus/CAE Usage: Specifying the surface for the normal velocity output is not supported Abaqus/CAE.

Local Normal Stiffness Output In Eigenvalue Extraction Analysis

In the engineering design of structures such as airplanes and automobiles it is common to estimate how a certain panel or shell structure locally resists the normal loads. This allows you to indicate weak spots and optimize the design.

The local normal stiffness is defined as a ratio of the normal force and normal displacement magnitudes at a surface node, where the force is applied statically at the surface node in the surface normal direction and the displacement is obtained from the linear static analysis of the model loaded with that single normal force (see *Figure 2*).

Calculation of the local normal stiffness for a panel requires solving a linear static problem for every node at the panel surface. In practical simulations (when the number of surface nodes is in the hundreds of thousands or millions), this calculation becomes extremely intensive and is not practical. However, you can calculate the approximate local normal stiffness very effectively by using the mode superposition method.

Abaqus/Standard can provide local normal stiffness field output at nodes of the element-based surface in the eigenvalue extraction analysis. The local normal stiffness field output is associated with the last output frame in the eigenvalue extraction procedure.

The output variable for the nodal local normal stiffness is STIFN.

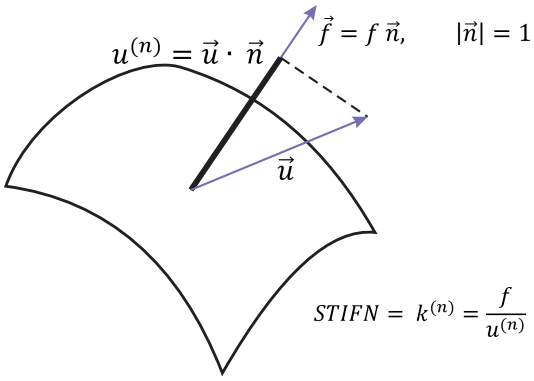


Figure 2: Local normal stiffness is calculated as a ratio of the normal force and the normal displacement.

Input File Usage: Use the following option to specify the element-based surface for the local normal stiffness output:

**NODE OUTPUT*, SURFACE=*element_based_surface_name*
STIFN

Abaqus/CAE Usage: Specifying the surface for the local normal stiffness output is not supported in Abaqus/CAE.

Parallel Execution of Steady-State Dynamic Analysis

Calculation of the frequency response for large finite element systems can be time consuming. Parallel execution is a practical option for steady-state dynamic analyses with millions of degrees of freedom, thousands of frequencies, and hundreds of load cases. For more information, see *Parallel Execution of Steady-State Dynamic Analyses*.

Fourier Transform in Linear Dynamics Procedures

Introduction

The direct and inverse Fourier transforms are available in Abaqus dynamic analysis procedures using the mode superposition method; that is, in the mode-based steady-state dynamic analysis procedure and in the transient modal dynamic analysis procedure. You can use the direct Fourier transform to convert the modal transient response into the modal frequency response, and you can convert the modal frequency response into the modal transient response using the inverse Fourier transform.

The direct and inverse Fourier transforms are defined by the following equations, respectively:

$$\hat{u}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} u(t)\, dt,$$

$$u(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} \hat{u}(\omega)\, d\omega,$$

where $\omega = 2\pi f$, f is the frequency, $u(t)$ is the transient response in the time domain, $\hat{u}(\omega)$ is the Fourier transform of the transient response; or $\hat{u}(\omega)$ is the frequency response in the frequency domain, and $u(t)$ is the inverse Fourier transform of it.

To clarify how the Fourier transform is used in Abaqus, consider how the linear dynamic analyses are performed. *Figure 3* schematically shows the conventional time domain and frequency domain mode-based analyses when the Fourier transform is not used.

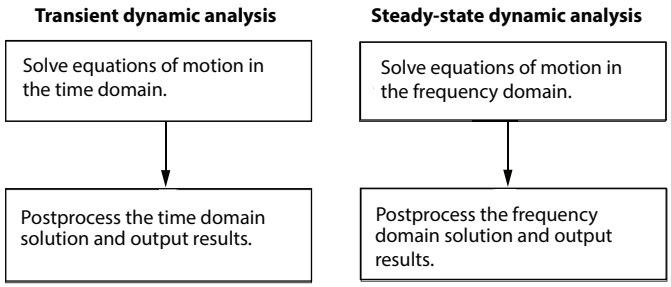


Figure 3: Conventional time domain and frequency domain mode-based analyses.

The units of the responses in the time and frequency domains calculated for the same model are the same.

Using the Fourier transform allows you to solve in the time domain but to output results in the frequency domain and vice versa. *Figure 4* illustrates options for time domain and frequency domain mode-based analyses using the Fourier transform.

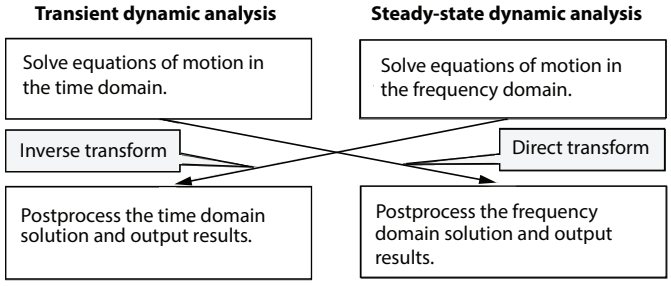


Figure 4: Time domain and frequency domain mode-based analyses using the Fourier transform.

The units of the Fourier transform of the transient response are not the same as the units of the frequency response calculated directly in the frequency domain for applied harmonic loads. Similarly, the units of the inverse Fourier

transform of the frequency response are not the same as the units of the transient response directly calculated in the time domain.

Using [ ] to indicate the dimension of a physical quantity (e.g., [T] denotes the dimension of time), we have the relationships:

$$[\textit{Direct Fourier transform of the transient response}] = [\textit{Frequency response}] \cdot [T]$$

$$\left[\textit{Inverse Fourier transform of the frequency response}\right] = \frac{[\textit{Transient response}]}{[T]}$$

Therefore, the workflows using the classical continuous Fourier transform are typically interesting when the dynamic excitation is presented by the direct Fourier transform of the loading history (in the time domain), or by the inverse Fourier transform of the harmonic loads defined in the frequency domain. In such cases, after the Fourier transform the solution has the appropriate physical dimensions (e.g., displacements would have the dimension of length) and can be used for postprocessing. Abaqus supports the continuous Fourier transform calculation. In addition, Abaqus also supports the discrete Fourier transform, which preserves the appropriate physical dimensions for various quantities, allowing you to manipulate the time domain and frequency domain responses.

In many practical cases, the units of the continuous Fourier transform can be adjusted to the frequency response units by multiplying by a constant factor. Abaqus provides this scaling option for experienced users.

Specifying the Direct Fourier Transform

The direct Fourier transform is utilized in the following pair of modal dynamic and mode-based steady-state dynamic steps. The steady-state dynamics step transforms the time-domain modal dynamic solution to a steady state frequency domain solution.

Input File Usage: Use the following options to request the direct Fourier transform calculation for the pair of modal analysis steps:

**STEP*
**MODAL DYNAMIC*
...
**END STEP*
**STEP*
**STEADY STATE DYNAMICS*, MODE BASED=FOURIER
...
**END STEP*

Abaqus/CAE Usage: Specifying the Fourier transform is not supported in Abaqus/CAE.

Specifying the Inverse Fourier Transform

The inverse Fourier transform is utilized in the following pair of steady-state dynamics and modal dynamic steps. The modal dynamic step transforms the frequency-domain steady-state dynamics solution to a time domain modal dynamics solution.

Input File Usage: Use the following options to request the inverse Fourier transform calculation for the pair of modal analysis steps:

**STEP*
**STEADY STATE DYNAMICS*
...
**END STEP*
**STEP*

*MODAL DYNAMIC, SOLVER=FOURIER
...
*END STEP

Abaqus/CAE Usage: Specifying the Fourier transform is not supported in Abaqus/CAE.

Selecting the Load Case for the Inverse Fourier Transform Calculation

The steady-state dynamics step can have multiple load cases. In such cases, you can select one load case for the inverse Fourier transform calculation. By default, Abaqus uses the first load case in the steady-state dynamics step.

Input File Usage: Use the following options to select a particular load case in the steady-state dynamics step for the inverse Fourier transform calculation:

*STEP
*STEADY STATE DYNAMICS
...
*LOAD CASE,NAME=*load_case_name1*
load_definitions
*END LOAD CASE
*LOAD CASE,NAME=*load_case_name2*
load_definitions
*END LOAD CASE
*END STEP
*STEP
*MODAL DYNAMIC, SOLVER=FOURIER, SSD LOAD CASE=*load_case_name*
...
*END STEP

Abaqus/CAE Usage: Specifying the Fourier transform is not supported in Abaqus/CAE.

Continuous Fourier Transform

Abaqus supports two methods for the Fourier integral calculation. In the first method the response under the integral is assumed to be a piecewise linear function between the time or frequency points for the calculation of the direct or inverse Fourier transform, respectively, as shown below. The products of the piecewise linear functions and the complex exponents are integrated analytically.

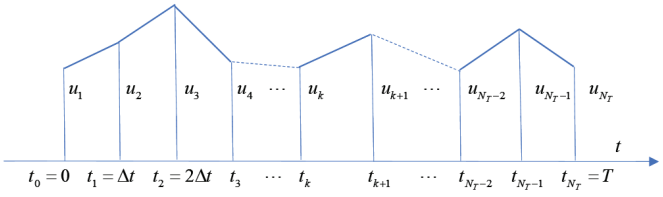


Figure 5: Piecewise linear approximation of the transient response for the direct Fourier transform calculation.

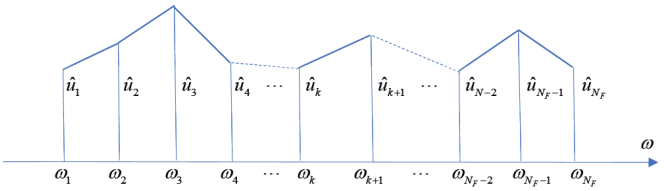


Figure 6: Piecewise linear approximation of the frequency response for the inverse Fourier transform calculation.

The second method assumes the integrand to be a piecewise constant function, and integration is replaced by summation of these function values scaled with the interval lengths in the time or frequency domains.

Input File Usage: Use the following option to specify the linear method for the Fourier transform calculation:

**FOURIER TRANSFORM CONTROLS*, METHOD=LINEAR

Use the following option to specify the constant method for the Fourier transform calculation:

**FOURIER TRANSFORM CONTROLS*, METHOD=CONSTANT

Use the following option to specify the scaling factor for the Fourier transform.

**FOURIER TRANSFORM CONTROLS*, SCALE=*value*

Abaqus/CAE Usage: Fourier transform is not supported in Abaqus/CAE.

Discrete Fourier transform

The "discrete Fourier transform," as it is implemented in Abaqus, is actually not the classical continuous Fourier transform or its approximation. The best definition of what is called the discrete Fourier transform" in Abaqus is the manipulation with the Fourier series. The direct discrete Fourier transform converts the transient response into the frequency response that is presented by harmonic oscillations at the given set of frequencies. On the other hand, the inverse discrete Fourier transform calculates the transient response as the superposition of harmonic oscillations at given frequencies. Both direct and inverse discrete Fourier transforms preserve the response units.

Input File Usage: Use the following option to specify the discrete method for the Fourier transform calculation:

**FOURIER TRANSFORM CONTROLS*, METHOD=DISCRETE

Abaqus/CAE Usage: Fourier transform is not supported in Abaqus/CAE.

Implicit Dynamic Analysis Using Direct Integration

Products: Abaqus/Standard Abaqus/CAE

Overview

A direct-integration dynamic analysis in Abaqus/Standard:

- must be used when nonlinear dynamic response is being studied;
- can be fully nonlinear (general dynamic analysis) or can be based on the modes of the linear system (subspace projection method); and
- can be used to study a variety of applications, including:
  - dynamic responses requiring transient fidelity and involving minimal energy dissipation;
  - dynamic responses involving nonlinearity, contact, and moderate energy dissipation; and
  - quasi-static responses in which considerable energy dissipation provides stability and improved convergence behavior for determining an essentially static solution.

References:

- *Defining an Analysis*
- *About Dynamic Analysis Procedures*
- **DYNAMIC*
- *Configuring a dynamic, implicit procedure*

General Dynamic Analysis

General nonlinear dynamic analysis in Abaqus/Standard uses implicit time integration to calculate the transient dynamic or quasi-static response of a system. The procedure can be applied to a broad range of applications calling for varying numerical solution strategies, such as the amount of numerical damping required to obtain convergence and the way in which the automatic time incrementation algorithm proceeds through the solution. Typical dynamic applications fall into three categories:

- Transient fidelity applications, such as an analysis of satellite systems, require minimal energy dissipation. In these applications small time increments are taken to accurately resolve the vibrational response of the structure, and numerical energy dissipation is kept at a minimum. These stringent requirements tend to degrade convergence behavior for simulations involving contact or nonlinearities.
- Moderate dissipation applications encompass a more general range of dynamic events in which a moderate amount of energy is dissipated by plasticity, viscous damping, or other effects. Typical applications include various insertion, impact, and forming analyses. The response of these structures can be either monotonic or nonmonotonic. Accurate resolution of high-frequency vibrations is usually not of interest in these applications. Some numerical energy dissipation tends to reduce solution noise and improve convergence behavior in these applications without significantly degrading solution accuracy.
- Quasi-static applications are primarily interested in determining a final static response. These problems typically show monotonic behavior, and inertia effects are introduced primarily to regularize unstable behavior. For example, the statically unstable behavior might be due to temporarily unconstrained rigid body modes or “snap-through” phenomena. Large time increments are taken when possible to obtain the final solution at

minimal computational cost. Considerable numerical dissipation might be required to obtain convergence during certain stages of the loading history.

An example of a transient fidelity application is available in *Modeling of an automobile suspension*. An analysis that includes both a moderate dissipation step and a quasi-static step is described in *Impact analysis of a pawl-ratchet device*.

Specifying the Application Type

Based on the classifications listed above, you should indicate the type of application you are studying when performing a general dynamic analysis. Abaqus/Standard assigns numerical settings based on your classification of the application type, and this classification can significantly affect a simulation. In some cases accurate results can be obtained with more than one application-type setting, in which case analysis efficiency should be considered. A general trend is that—among the three classifications—the high-dissipation quasi-static classification tends to result in the best convergence behavior and the low-dissipation transient fidelity classification tends to have the highest likelihood of convergence difficulty.

Input File Usage: Use the following option for transient fidelity applications:

**DYNAMIC*, APPLICATION=TRANSIENT FIDELITY (default for models without contact)

Use the following option for moderate dissipation applications:

**DYNAMIC*, APPLICATION=MODERATE DISSIPATION (default for models with contact)

Use the following option for quasi-static applications:

**DYNAMIC*, APPLICATION=QUASI-STATIC

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Implicit**

The application type is specified in the **Edit Step** dialog box:

Basic: **Application: Transient fidelity**, **Moderate dissipation**, **Quasi-static**, or **Analysis product default**

Diagnostics for Modeling Errors Associated with Mass Properties

Accurate representation of inertia properties is necessary for accurate dynamic analyses. In some cases Abaqus/Standard provides diagnostic messages when it detects likely modeling errors associated with the specification of inertia properties. The most common way of specifying inertia properties is with material densities. Abaqus/Standard issues a warning message to the data (`.dat`) file if a material density is omitted in a dynamic analysis (this warning is not issued if the density is zero only for certain values of temperature or field variables). Other methods of specifying inertia properties include:

- point mass and rotary inertia definitions, and
- constraining nodes without inertia themselves to nodes having inertia properties defined.

In some circumstances Abaqus/Standard attempts to solve systems of equations involving effective inversion of the global mass matrix to directly adjust velocities and accelerations during a general dynamic analysis as described in *Initial Conditions* and *Intermittent Contact/Impact* below. These additional velocity and acceleration adjustments occur by default only for transient fidelity application types as defined above. If the global mass

matrix is found to be singular, Abaqus/Standard issues an error message by default, because singular mass is an indication that the mass properties are not realistic due to a modeling error.

Diagnostic feedback specific to the global mass matrix being singular is typically not provided for quasi-static and moderate dissipation application types, although warnings typically are issued regarding the lack of material density. Singular mass is not necessarily detrimental to a quasi-static analysis. For example, it would be reasonable to only define inertia properties (such as density) in components or regions with temporary static instabilities (such as initially unconstrained rigid body modes that become constrained once contact occurs) in a quasi-static analysis.

You can control the course of action Abaqus/Standard takes on detecting a singular global mass matrix.

Input File Usage: Use the following default option to issue an error message and stop execution if a singular global mass matrix is detected when calculating velocity and acceleration adjustments:

**DYNAMIC*, SINGULAR MASS=ERROR

Use the following option to issue a warning message and avoid velocity and acceleration adjustments (that is., continue time integration using current velocities and accelerations) if a singular global mass matrix is detected:

**DYNAMIC*, SINGULAR MASS=WARNING

Use the following option to adjust velocities and accelerations even if a singular mass matrix is detected. This setting can result in large, non-physical velocity and/or acceleration adjustments, which can, in turn, cause poor time integration solutions and artificial convergence difficulties. *This approach is not generally recommended; it should be used only in special cases when the analyst has a thorough understanding of how to interpret results obtained in this way.*

**DYNAMIC*, SINGULAR MASS=MAKE ADJUSTMENTS

Abaqus/CAE Usage: The default singular mass setting cannot be modified in Abaqus/CAE.

Numerical Details

The effect of the application-type classification on numerical aspects of general dynamic analyses is described below. In most cases the settings determined by the application type are sufficient to successfully perform an analysis. However, detailed user controls are provided to override settings on an individual basis.

Time Integration Methods

Abaqus/Standard uses the Hilber-Hughes-Taylor time integration by default unless you specify that the application type is quasi-static. The Hilber-Hughes-Taylor operator is an extension of the Newmark β-method. Numerical parameters associated with the Hilber-Hughes-Taylor operator are tuned differently for moderate dissipation and transient fidelity applications (as discussed later in this section). The backward Euler operator is used by default if the application classification is quasi-static.

These time integration operators are implicit, which means that the operator matrix must be inverted and a set of simultaneous nonlinear dynamic equilibrium equations must be solved at each time increment. This solution is done iteratively using Newton's method. The principal advantage of these operators is that they are unconditionally stable for linear systems; there is no mathematical limit on the size of the time increment that can be used to integrate a linear system. An unconditionally stable integration operator is of great value when

studying structural systems because a conditionally stable integration operator (such as that used in the explicit method) can lead to impractically small time steps and, therefore, a computationally expensive analysis.

Marching through a simulation with a finite time increment size generally introduces some degree of numerical damping. This damping differs from the material damping discussed in *Material Damping* (and in many cases these two forms of damping work well together). The amount of damping associated with the time integration varies among the operator types (for example, the backward Euler operator tends to be more dissipative than the Hilber-Hughes-Taylor operator) and in many cases (such as with the Hilber-Hughes-Taylor operator) depends on settings of numerical parameters associated with the operator. The ability of the operator to effectively treat contact conditions is often of considerable importance with respect to their usefulness. For example, some changes in contact conditions can result in "negative damping" (nonphysical energy source) for many time integrators, which can be very undesirable.

It is possible to override the time integrator implied by the application-type classification; for example, you can perform a moderate dissipation dynamic analysis using the backward Euler integrator. Changing the default integrator is not generally recommended but might be useful in special cases.

Input File Usage: Use the following option to use the Hilber-Hughes-Taylor integrator with default integrator parameter settings corresponding to those for transient fidelity applications:

**DYNAMIC*, TIME INTEGRATOR=HHT-TF

Use the following option to use the Hilber-Hughes-Taylor integrator with default integrator parameter settings corresponding to those for moderate dissipation applications:

**DYNAMIC*, TIME INTEGRATOR=HHT-MD

Use the following option to use the backward Euler integrator:

**DYNAMIC*, TIME INTEGRATOR=BWE

Abaqus/CAE Usage: The default time integrator cannot be modified in Abaqus/CAE.

Additional Control over Integrator Parameters

Additional user controls enable modifications to settings of numerical parameters associated with the Hilber-Hughes-Taylor operator (see *Hilber, Hughes, and Taylor (1977)* for descriptions of the numerical parameters). The default parameter settings depend on the specified application type, as indicated in *Table 1* (see *Czekanski, El-Abbasi, and Meguid (2001)* for the basis of these settings).

Table 1: Default parameters for the Hilber-Hughes-Taylor integrator.

Parameter	Application	
	Transient Fidelity	Moderate Dissipation
α	–0.05	–0.41421
β	0.275625	0.5
γ	0.55	0.91421

These parameters can be adjusted or modified individually if the Hilber-Hughes-Taylor operator is being used. If the default settings of these parameters correspond to the transient fidelity settings shown in *Table 1* and you

explicitly modify the α parameter alone, the other parameters are adjusted automatically to $\beta = 1/4 \cdot (1 - \alpha)^2$ and $\gamma = 1/2 - \alpha$. This relation provides control of the numerical damping associated with the time integrator while preserving desirable characteristics of the integrator. The numerical damping grows with the ratio of the time increment to the period of vibration of a mode. Negative values of α provide damping; whereas $\alpha = 0$ results in no damping (energy preserving) and is exactly the trapezoidal rule (sometimes called the Newmark β-method, with $\beta = 1/4$ and $\gamma = 1/2$). The setting $\alpha = -1/3$ provides the maximum numerical damping. It gives a damping ratio of about 6% when the time increment is 40% of the period of oscillation of the mode being studied. Allowable values of α, β, and γ are: $-\frac{1}{2} \le \alpha \le 0$, $\beta > 0$, $\gamma \ge \frac{1}{2}$.

Input File Usage: **DYNAMIC*, ALPHA=α, BETA=β, GAMMA=γ

Abaqus/CAE Usage: Only the α parameter can be modified in Abaqus/CAE:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Alpha: Specify:** α

Default Incrementation Schemes

Automatic time incrementation is used by default for nonlinear dynamic procedures. The main factors used to control adjustments to the time increment size for an implicit dynamic procedure are the convergence behavior of the Newton iterations and the accuracy of the time integration. The time increment size might vary considerably during an analysis. Details of the time increment control algorithm depend on the type of dynamic application you are studying.

The following factors are considered by default in the time increment control algorithm if you specify a quasi-static–type application (the same factors control the time increment size for purely static analyses):

- The time increment size is reduced if an increment appears to be diverging or if the convergence rate is slow.
- The time increment size is fairly aggressively increased if rapid convergence occurs in previous increments.

Analyses for moderate dissipation-type applications also use these same factors, as well as a default upper bound on the time increment size equal to one tenth of the step duration.

The following factors are considered by default in the time increment control algorithm if you specify a transient fidelity–type application:

- The time increment size is reduced if an increment appears to be diverging or if the convergence rate is slow.
- The time increment size is reduced if changes in contact status are detected during the first attempt of processing an increment. The new increment size is set such that the end of the increment corresponds to the average time of the contact status changes that were detected with the previous increment size. (In such cases an additional very small time increment is used to enforce compatibility of velocities and accelerations across active contact interfaces.)
- The time increment size is reduced if the half-increment residual (out-of-balance force) halfway through a time increment exceeds the half-increment residual tolerance, which is 10,000 times the time average force for a contact analysis or 1000 times the time average force for an analysis without contact.
- The time increment is gradually increased if rapid convergence occurs in previous increments.
- The upper bound for the time increment size is equal to 1/100 of the step duration.

Intermittent Contact/Impact

The second and third factors described in the preceding list often result in very small time increment sizes for contact simulations that are performed as a transient fidelity application (and the time increment size tends to remain small due to the fourth factor). This problem can be avoided by specifying a different application type or by using more detailed user controls, as discussed below.

General Settings for the Time Increment Controls

A high-level user control over which factors are considered by the time increment control algorithm can be used to override the defaults implied by the specified application type for the analysis. Regardless of the application type you have specified, you can enforce time increment controls associated with either quasi-static applications or transient fidelity applications.

Input File Usage: Use the following option to obtain the aggressive time increment control settings associated with quasi-static applications:

**DYNAMIC*, INCREMENTATION=AGGRESSIVE

Use the following option to obtain the more conservative time increment control settings associated with transient fidelity applications:

**DYNAMIC*, INCREMENTATION=CONSERVATIVE

Abaqus/CAE Usage: The default time incrementation control settings cannot be modified in Abaqus/CAE.

Controlling the Half-Increment Residual

Controls associated with the half-increment residual tolerance are provided for tuning the time incrementation. These controls are intended for advanced users and typically do not need to be modified.

Input File Usage: Use the following option to specify that no check of the half-increment residual should be performed:

**DYNAMIC*, NOHAF

Use the following option to specify the half-increment residual tolerance as a scale factor of the time average force (moment):

**DYNAMIC*, HALFINC SCALE FACTOR=*scale factor*

Use the following option to directly specify the half-increment residual force tolerance (the half-increment residual moment tolerance is the half-increment residual force tolerance times the characteristic element length automatically calculated):

**DYNAMIC*, HAFTOL=*tolerance*

Abaqus/CAE Usage: Use the following option to specify that no check of the half-increment residual should be performed:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: toggle on **Suppress half-increment residual calculation**

Use the following option to specify the half-increment residual tolerance as a scale factor of the time average force (moment):

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Half-increment Residual**: **Specify scale factor:** *scale factor*

Use the following option to specify the half-increment residual force tolerance directly:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Half-increment Residual: Specify value:** *tolerance*

Controlling Incrementation Involving Contact

By default, specifying a transient fidelity application typically results in reduced time increment sizes on changes in contact status. An extra time increment with a very small size is subsequently performed to enforce compatibility of velocities and accelerations across active contact interfaces. Direct user control over these incrementation aspects is available.

Input File Usage: Use the following option to avoid automatically cutting back the increment size and enforcing velocity and acceleration compatibility in the contact region on changes in contact status:

**DYNAMIC*, IMPACT=NO

Use the following option to automatically cut back the increment size and enforce velocity and acceleration compatibility in the contact region on changes in contact status:

**DYNAMIC*, IMPACT=AVERAGE TIME

Use the following option to enforce velocity and acceleration compatibility in the contact region without automatically cutting back the increment size on changes in contact status:

**DYNAMIC*, IMPACT=CURRENT TIME

Abaqus/CAE Usage: The default contact incrementation scheme cannot be modified in Abaqus/CAE.

Direct Time Incrementation

You may directly specify the time increment size to be used. This approach is not generally recommended but may be useful in special cases. The analysis ends if convergence tolerances are not satisfied within the maximum number of iterations allowed.

It is possible to ignore convergence tolerances: the solution to an increment is accepted after the specified maximum number of iterations allowed even if convergence tolerances are not satisfied. Ignoring convergence tolerances can result in highly nonphysical results and is not recommended except by analysts with a thorough understanding of how to interpret results obtained this way.

Input File Usage: Use the following option to directly specify the time increment:

**DYNAMIC*, DIRECT

Use the following option to ignore convergence tolerances after the maximum number of iterations is reached:

**DYNAMIC*, DIRECT=NO STOP

Abaqus/CAE Usage: Use the following option to specify the time increment directly:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Fixed**

Use the following option to ignore convergence tolerances after the maximum number of iterations is reached:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Accept solution after reaching maximum number of iterations**

Default Amplitude for Loads

Loads such as applied forces or pressures are ramped on by default if you have selected the quasi-static application classification; such ramping tends to enhance robustness because the load increment size is proportional to the time increment size. For example, if the Newton iterations are unable to converge for a particular time increment size, the automatic time incrementation algorithm reduces the time increment size and restart the Newton iterations with a smaller load increment considered.

For the other application classifications the dynamic procedure applies loads with a step function by default such that the full load is applied in the first increment of the step (regardless of the time increment size) and the load magnitude remains constant over each step. Thus, if the first increment is unable to converge with the original time increment size, reducing the time increment does not reduce the load increment by default. In some cases the convergence behavior still improves on reducing the time increment because the regularizing effect of inertia on the integration operators is inversely proportional to the square of the time increment size. See *Defining an Analysis* for more information on default amplitude types for the various procedures and how to override the default.

The "Subspace Projection" Method

The alternative approach provided in Abaqus/Standard for nonlinear dynamic problems is the "subspace projection" method. See *Subspace dynamics* for the theory behind this method. In this method the modes of the linear system are extracted in an eigenfrequency extraction step (*Natural Frequency Extraction*) prior to the dynamic analysis and are used as a small set of global basis vectors to develop the solution. These modes include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The method works well when the system exhibits mildly nonlinear behavior, such as small regions of plastic yielding or rotations that are not small but not too large.

This method can be very effective. As with the other direct integration methods, it is more expensive in terms of computer time than the modal methods of purely linear dynamic analysis, but it is often significantly less expensive than the direct integration of all of the equations of motion of the model. However, since the subspace projection method is based on the modes of the system, it is not accurate if there is extreme nonlinear response that cannot be modeled well by the modes that form the basis of the solution.

Input File Usage: **DYNAMIC*, SUBSPACE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Subspace**

Selecting the Modes on Which to Project

You can select the modes of the system on which the subspace projection is performed. The mode numbers can be listed individually, or they can be generated automatically. If you choose not to select the modes, all modes extracted in the prior frequency extraction step, including residual modes if they were activated, are used in the subspace projection.

Input File Usage: Use one of the following options:

**SELECT EIGENMODES*
**SELECT EIGENMODES*, GENERATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Subspace**: **Basic**: **Number of modes to use**: **All** or **Specify**

Numerical Implementation

The subspace projection method is implemented in Abaqus/Standard using the explicit (central difference) operator to integrate the equations of motion written in terms of the modes of the linear system. This integration method is particularly effective here because the modes are orthogonal with respect to the mass matrix so that the projected system always has a diagonal mass matrix.

A fixed time increment is used: this increment is the smaller of the time increment that you specify or 80% of the stable time increment, which is $2/\omega_{max}$ for the linear system, where ω_{max} is the highest circular frequency of the modes that are used as the basis of the solution. The 80% factor is intended as a safety factor so that any increase in this highest frequency caused by nonlinear effects is less likely to cause the integration to become unstable. The 80% is rather arbitrary; in some cases it might be nonconservative. You must monitor the response—for example, the energy balance—to ensure that the time increment is not causing instability. Instability is a concern if the nonlinearities can stiffen the system significantly, although in many practical cases such stiffening effects are more prominent in increasing the lower frequencies of the system than in affecting the highest frequencies that are likely to be retained to represent the dynamic behavior accurately.

Accuracy of the Subspace Projection Method

The effectiveness of the subspace projection method depends on the value of the modes of the linear system as a set of global interpolation functions for the problem, which is a matter of judgment on your part—the same sort of judgment as required when deciding if a particular mesh of finite elements is sufficient. The method is valuable for mildly nonlinear systems and for cases where it is easy to extract enough modes that you can be confident that they describe the system adequately.

If nonlinear geometric effects are considered in the subspace dynamics step, it is possible to perform a dynamic simulation for some time, reextract the modes on the current stressed geometry by using another frequency extraction step, and then continue the analysis with the new modes as the subspace basis system. This procedure can improve the accuracy of the method in some cases.

Material Damping

You can introduce Rayleigh damping, as explained in *Material Damping*. This damping acts in addition to numerical damping associated with the time integrator (discussed previously).

Input File Usage: **DAMPING*, ALPHA=α_R, BETA=β_R

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damping**: **Alpha** and **Beta**

Controlling Rate-Dependent Yield and Friction

You can control whether to consider or ignore the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient within the step in a quasi-static response.

Input File Usage: Use the following option to consider rate dependence within the step (default):

**DYNAMIC*, APPLICATION=QUASI-STATIC, RATE DEPENDENCE=ON

Use the following option to ignore rate dependence within the step:

**DYNAMIC*, APPLICATION=QUASI-STATIC, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient.

Initial Conditions

Initial Conditions describes all the available initial conditions. Initial velocities must be defined in global directions regardless of the use of nodal transformations (see *Transformed Coordinate Systems*).

If initial velocities are specified at nodes for which displacement boundary conditions are also specified, the initial velocities are ignored at these nodes. However, if a displacement boundary condition refers to an amplitude curve with an analytically defined time variation (that is, excluding the piecewise linear tabular and equally spaced definitions), Abaqus/Standard computes the initial velocity for the nodes involved in the boundary condition as the time derivative (evaluated at time zero) of the analytic variation.

When initial velocities are specified for dynamic analysis, they should be consistent with all the constraints on the model, especially time-dependent boundary conditions. Abaqus/Standard ensures that initial velocities are consistent with boundary conditions and with multi-point and equation constraints but does not check for consistency with internal constraints such as incompressibility of the material. In case of a conflict, boundary conditions and multi-point constraints take precedence over initial conditions.

Specified initial velocities are used in a dynamic step only if it is the first dynamic step in an analysis. If a dynamic step is not the first dynamic step and there is an immediately preceding dynamic step, the velocities from the end of the preceding step are used as the initial velocities for the current step. If a dynamic step is not the first dynamic step and the immediately preceding step is not a dynamic step, zero initial velocities are assumed for the current step.

Controlling Calculation of Accelerations at the Beginning of a Dynamic Step

By default, Abaqus/Standard calculates accelerations at the beginning of the dynamic step for transient fidelity applications. You can choose to bypass these acceleration calculations, in which case Abaqus/Standard assumes that initial accelerations for the current step are zero unless there is an immediately preceding dynamic step. If the immediately preceding step is also a dynamic step, bypassing the acceleration calculations causes Abaqus/Standard to use the accelerations from the end of the previous step to continue the new step. It is appropriate to bypass the acceleration calculations if the loading has not changed suddenly at the start of the dynamic step, but it is not correct if the loading at the beginning of the first increment is significantly different from that at the end of the previous step. In cases where large loads are applied suddenly, high-frequency noise due to the bypass of the acceleration calculations might greatly increase the half-increment residual.

Input File Usage: **DYNAMIC*, INITIAL=NO

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Initial acceleration calculations at beginning of step: Bypass**

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6), to warping degree of freedom 7 in open-section beam elements, to fluid pressure degree of freedom 8 for hydrostatic fluid elements, or to acoustic pressure degree of freedom 8 for acoustic elements (*Boundary Conditions*).

Amplitude references can be used to prescribe time-varying boundary conditions in a direct-integration dynamic step. Default amplitude variations are described in *Defining an Analysis*.

In direct time integration dynamic analysis, when a node with a prescribed motion is used in an equation constraint or a multi-point constraint to control the motion of another node, the equation or multi-point constraint is imposed correctly for the displacement and velocity of the dependent node. However, the acceleration is not rigorously transmitted to the dependent node, which might cause some high-frequency noise.

In the subspace projection method, it is not possible to specify nonzero boundary conditions directly. Instead, acceleration boundary conditions can be approximated by using appropriate combinations of large point masses and concentrated loads. At the node where such a boundary condition is desired, attach a large point mass that is approximately 10^5–10^6 times larger than the mass of the original model. In addition, a concentrated load of magnitude equal to the product between the large point mass and the desired acceleration must be specified in the direction of the approximated boundary condition. Since the point mass is significantly larger than the mass of the model, the big mass–concentrated load combination approximates the desired acceleration in the specified direction accurately. Boundary conditions other than accelerations must be converted into acceleration histories before they can be approximated.

Loads

The following loads can be prescribed in a dynamic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.
- Distributed pressure or volumetric accelerations (on acoustic elements) can be applied; these are described in *Coupled Acoustic-Structural Analysis*.

Predefined Fields

The following predefined fields can be specified in a dynamic analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in stress/displacement elements, nodal temperatures can be specified as a predefined field. Any difference between the applied and initial temperatures causes thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.

Material Options

Most material models that describe mechanical behavior are available for use in a dynamic analysis. The following material properties are not active during a dynamic analysis: thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, and pore fluid flow properties.

Rate-dependent material properties (*Time Domain Viscoelasticity*, *Hysteresis in Elastomers*, *Rate-Dependent Yield*, and *Two-Layer Viscoplasticity*) can be included in a dynamic analysis.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature, pressure, and electrical potential degrees of freedom) can be used in a dynamic analysis. Inertia effects are ignored in hydrostatic fluid elements, and the inertia of the fluid in pore pressure elements is not taken into account.

Output

In addition to the usual output variables available in Abaqus/Standard (see *Abaqus/Standard Output Variable Identifiers*), the following variables are provided specifically for implicit dynamic analysis:

Variables for a specified element set or for the entire model:

XC Current coordinates of the center of mass.

XC*n* Coordinate *n* of the center of mass ($n = 1, 2, 3$).

UC Displacement of the center of mass.

UC*n* Displacement component *n* of the center of mass ($n = 1, 2, 3$).

URC*n* Rotation component *n* of the center of mass.

VC Equivalent rigid body velocity components.

VC*n* Component *n* of the equivalent rigid body velocity ($n = 1, 2, 3$).

VRC*n* Component *n* of the equivalent rigid body angular velocity ($n = 1, 2, 3$).

HC Angular momentum about the center of mass.

HC*n* Component *n* of the angular momentum about the center of mass ($n = 1, 2, 3$).

HO Angular momentum about the origin.

HO*n* Component *n* of the angular momentum about the origin ($n = 1, 2, 3$).

RI Rotary inertia about the origin.

RI*ij*
: ij-component of the rotary inertia about the origin ($i \leq j \leq 3$).

MASS
: Mass.

VOL
: Current volume.

Input File Template

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**
*STEP (,NLGEOM)
Once NLGEOM is specified, it will be active in all subsequent steps.
*DYNAMIC
Data line to control automatic time incrementation
*BOUNDARY
Data lines to describe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD and/or *INCIDENT WAVE
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to prescribe predefined fields
*CECHARGE and/or *DECHARGE (if electrical potential degrees of
freedom are active)
Data lines to specify charges
*END STEP
```

References

- Czekanski, A., N. El-Abbasi, and S. A. Meguid, “Optimal Time Integration Parameters for Elastodynamic Contact Problems,” *Communications in Numerical Methods in Engineering*, vol. 17, pp. 379–384, 2001.
- Hilber, H. M., T. J. R. Hughes, and R. L. Taylor, “Improved Numerical Dissipation for Time Integration Algorithms in Structural Dynamics,” *Earthquake Engineering and Structural Dynamics*, vol. 5, pp. 283–292, 1977.

Explicit Dynamic Analysis

Products: Abaqus/Explicit Abaqus/CAE

Overview

An explicit dynamic analysis:

- is computationally efficient for the analysis of large models with relatively short dynamic response times and for the analysis of extremely discontinuous events or processes;
- allows for the definition of very general contact conditions (*About Contact Interactions*);
- uses a consistent, large-deformation theory—models can undergo large rotations and large deformation;
- can use a geometrically linear deformation theory—strains and rotations are assumed to be small (see *Defining an Analysis*);
- can be used to perform an adiabatic stress analysis if inelastic dissipation is expected to generate heat in the material (see *Adiabatic Analysis*);
- can be used to perform quasi-static analyses with complicated contact conditions; and
- allows for either automatic or fixed time incrementation to be used—by default, Abaqus/Explicit uses automatic time incrementation with the global time estimator.

References:

- *Defining an Analysis*
- **DYNAMIC*
- *Configuring a dynamic, explicit procedure*

Explicit Dynamic Procedure

The explicit dynamics procedure performs a large number of small time increments efficiently. An explicit central-difference time integration rule is used; each increment is relatively inexpensive (compared to the direct-integration dynamic analysis procedure available in Abaqus/Standard) because there is no solution for a set of simultaneous equations. The explicit central-difference operator satisfies the dynamic equilibrium equations at the beginning of the increment, t; the accelerations calculated at time t are used to advance the velocity solution to time $t + \Delta t/2$ and the displacement solution to time $t + \Delta t$.

Input File Usage: **DYNAMIC*, EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**

Central-Difference Time Integration with Implicit Corrections

The explicit dynamic analysis procedure is based on the implementation of an explicit integration rule together with the use of diagonal (“lumped”) element mass matrices. A correction phase is used to implicitly treat connectors and constraints.

Predicted nodal accelerations, $\ddot{u}^{N\ Pred}_{(i)}$, at the beginning of increment number *i* of an explicit dynamic step are computed by:

$$\ddot{u}^{N\ Pred}_{(i)} = \left(M^{NJ}\right)^{-1}\left(P^{J}_{(i)} - I^{J}_{(i)}\right),$$

where M^{NJ} is the mass matrix, P^{J} is the applied load vector, and I^{J} is the explicit internal force vector. A lumped mass matrix is used because its inverse is simple to compute and because the vector multiplication of the mass inverse by the inertial force requires only *n* operations, where *n* is the number of degrees of freedom in the model. The internal force vector, I^{J}, is assembled from contributions from the individual elements such that a global stiffness matrix need not be formed.

The equations of motion for the body are integrated using the explicit central-difference integration rule to compute predicted mid-increment nodal velocity (or angular velocity), $\dot{u}^{N\ Pred}_{\left(i+\frac{1}{2}\right)}$, and predicted end-of-increment displacement (or rotation), $u^{N\ Pred}_{(i+1)}$ as follows:

$$\dot{u}^{N\ Pred}_{\left(i+\frac{1}{2}\right)} = \dot{u}^{N}_{\left(i-\frac{1}{2}\right)} + \frac{\Delta t_{(i+1)}+\Delta t_{(i)}}{2}\ddot{u}^{N\ Pred}_{(i)},$$

$$u^{N\ Pred}_{(i+1)} = u^{N}_{(i)} + \Delta t_{(i+1)}\dot{u}^{N\ Pred}_{\left(i+\frac{1}{2}\right)}.$$

The central-difference integration operator is explicit in the sense that the predicted kinematic state is advanced using known values of $\dot{u}^{N}_{\left(i-\frac{1}{2}\right)}$ and $u^{N}_{(i)}$ from the previous increment.

However, the predicted kinematic quantities do not account for implicitly treated connections and constraints. To account for these entities, implicit forces are computed considering the predicted configuration. Calculation of the implicit force vector, $F_{Impl}{}^{J}_{(i)}$, often requires solution of nonlinear systems of equations each increment, which is typically efficient but can degrade performance for situations with overlapping constraints, as discussed in *Constraint Enforcement and Overlapping Constraints*. These implicit forces cause nodal acceleration corrections at increment *i*:

$$\ddot{u}^{N\ Correction}_{(i)} = \left(M^{NJ}\right)^{-1}\left(F_{Impl}{}^{J}_{(i)}\right).$$

Acceleration corrections are applied to determine corrected kinematic quantities that satisfy connection and constraint behavior as follows:

$$\ddot{u}^{N}_{(i)} = \ddot{u}^{N\ Pred}_{(i)} + \ddot{u}^{N\ Correction}_{(i)},$$

$$\dot{u}^{N}_{\left(i+\frac{1}{2}\right)} = \dot{u}^{N}_{\left(i-\frac{1}{2}\right)} + \frac{\Delta t_{(i+1)}+\Delta t_{(i)}}{2}\ddot{u}^{N}_{(i)},$$

$$u^{N}_{(i+1)} = u^{N}_{(i)} + \Delta t_{(i+1)}\dot{u}^{N}_{\left(i+\frac{1}{2}\right)}.$$

Sometimes it is necessary to enforce the constraints implicitly, especially if there are multiple constraints applied at a node. The implicit enforcement of constraints can lead to a performance degradation (see *Computational Cost for Constraint Solution in Abaqus/Explicit*).

Nodal Mass and Inertia

The explicit integration scheme in Abaqus/Explicit requires nodal mass or inertia to exist at all activated degrees of freedom (see *Conventions*) unless constraints are applied using boundary conditions. More precisely, a nonzero

nodal mass must exist unless all activated translational degrees of freedom are constrained and nonzero rotary inertia must exist unless all activated rotational degrees of freedom are constrained. Nodes that are part of a rigid body do not require mass, but the entire rigid body must possess mass and inertia unless constraints are used. Nodes that belong to Eulerian elements also do not require mass, since the surrounding Eulerian elements might be void at some time during the simulation.

When degrees of freedom at a node are activated by elements with a nonzero mass density (for example, solid, shell, beam) or mass and inertia elements, a nonzero nodal mass or inertia occurs naturally from the assemblage of lumped mass contributions.

When degrees of freedom at a node are activated by elements with no mass (for example, spring, dashpot, or connector elements), care must be taken either to constrain the node or to add mass and inertia as appropriate.

Stability

The explicit procedure integrates through time by using many small time increments. The central-difference operator is conditionally stable, and the stability limit for the operator (with no damping) is given in terms of the highest frequency of the system as

$$\Delta t \leq \frac{2}{\omega_{max}}.$$

With damping, the stable time increment is given by

$$\Delta t \leq \frac{2}{\omega_{max}} \left(\sqrt{1+\xi_{max}^2} - \xi_{max} \right),$$

where ξ_{max} is the fraction of critical damping in the mode with the highest frequency. Contrary to our usual engineering intuition, introducing damping to the solution reduces the stable time increment. In Abaqus/Explicit a small amount of damping is introduced in the form of bulk viscosity to control high frequency oscillations. Physical forms of damping, such as dashpots or material damping, can also be introduced. Bulk viscosity and material damping are discussed below.

Estimating the Stable Time Increment Size

An approximation to the stability limit is often written as the smallest transit time of a dilatational wave across any of the elements in the mesh

$$\Delta t \approx \frac{L_{min}}{c_d},$$

where L_{min} is the smallest element dimension in the mesh and c_d is the dilatational wave speed in terms of λ_0 and μ_0, defined below.

In general, for beams, conventional shells, and membranes the element thickness or cross-sectional dimensions are not considered in determining the smallest element dimension; the stability limit is based on the midplane or membrane dimensions only. When the transverse shear stiffness is defined for shell elements (see *Shell Section Behavior*), the stable time increment will also be based on the transverse shear behavior.

This estimate for Δt is only approximate and in most cases is not a conservative (safe) estimate. In general, the actual stable time increment chosen by Abaqus/Explicit will be less than this estimate by a factor between $1/\sqrt{2}$ and 1 in a two-dimensional model and between $1/\sqrt{3}$ and 1 in a three-dimensional model. The time increment chosen by Abaqus/Explicit also accounts for any stiffness behavior in a model associated with penalty contact. For further discussion, see *Computational Cost* below.

Stable Time Increment Report

Abaqus/Explicit writes a report to the status (`.sta`) file during the data check phase of the analysis that contains an estimate of the minimum stable time increment and a listing of the elements with the smallest stable time increments and their values. The initial stable time increments listed do not include damping (bulk viscosity), mass scaling, or penalty contact effects.

This listing is provided because often a few elements have much smaller stability limits than the rest of the elements in the mesh. The stable time increment can be increased by modifying the mesh to increase the size of the controlling element or by using appropriate mass scaling.

Dilatational Wave Speed

The current dilatational wave speed, c_d, is determined in Abaqus/Explicit by calculating the effective hypoelastic material moduli from the material's constitutive response. Effective Lamé's constants, $\widehat{\lambda}$ and $\widehat{G} = 2\widehat{\mu}$, are determined in the following manner. Define Δp as the increment in the mean stress, $\Delta \mathbf{S}$ as the increment in the deviatoric stress, $\Delta \epsilon_{vol}$ as the increment of volumetric strain, and $\Delta \mathbf{e}$ as the deviatoric strain increment. We assume a hypoelastic stress-strain rule of the form

$$\Delta p = \left(3\widehat{\lambda} + 2\widehat{\mu}\right) \Delta \epsilon_{vol},$$

$$\Delta \mathbf{S} = 2\widehat{\mu} \Delta \mathbf{e}.$$

The effective moduli can then be computed as

$$3\widehat{K} = 3\widehat{\lambda} + 2\widehat{\mu} = \frac{\Delta p}{\Delta \epsilon_{vol}},$$

$$2\widehat{\mu} = \frac{\Delta \mathbf{S} : \Delta \mathbf{e}}{\Delta \mathbf{e} : \Delta \mathbf{e}},$$

$$\widehat{\lambda} + 2\widehat{\mu} = \frac{1}{3}\left(3\widehat{K} + 4\widehat{\mu}\right).$$

For shell elements defined by a shell cross-section that requires numerical integration (see *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*), the effective moduli for the section are computed by integrating the effective moduli at the section points through the thickness. These effective moduli represent the element stiffness and determine the current dilatational wave speed in the element as

$$c_d = \sqrt{\frac{\widehat{\lambda} + 2\widehat{\mu}}{\rho}},$$

where ρ is the density of the material.

In an isotropic, elastic material the effective Lamé's constants can be defined in terms of Young's modulus, E, and Poisson's ratio, ν, by

$$\widehat{\lambda} = \lambda_0 = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

and

$$\widehat{\mu} = \mu_0 = \frac{E}{2(1+\nu)}.$$

Time Incrementation

The time increment used in an analysis must be smaller than the stability limit of the central-difference operator. Failure to use a small enough time increment will result in an unstable solution. When the solution becomes

unstable, the time history response of solution variables such as displacements will usually oscillate with increasing amplitudes. The total energy balance will also change significantly.

If the model contains only one material type, the initial time increment is directly proportional to the size of the smallest element in the mesh. If the mesh contains uniform size elements but contains multiple material descriptions, the element with the highest wave speed will determine the initial time increment.

In nonlinear problems—those with large deformations and/or nonlinear material response—the highest frequency of the model will continually change, which consequently changes the stability limit. Abaqus/Explicit has two strategies for time incrementation control: fully automatic time incrementation (where the code accounts for changes in the stability limit) and fixed time incrementation.

Scaling the Time Increment

To reduce the chance of a solution going unstable, you can adjust the stable time increment computed by Abaqus/Explicit by a constant scaling factor. This factor can be used to scale the default global time estimate, the element-by-element estimate, or the fixed time increment based on the initial element-by-element estimate; it cannot be used to scale a fixed time increment specified directly by you.

Input File Usage: Use the following option to scale the stable time increment based on the global time estimate:

**DYNAMIC*, EXPLICIT, SCALE FACTOR=*f*

Use the following option to scale the stable time increment based on the element-by-element estimate:

**DYNAMIC*, EXPLICIT, ELEMENT BY ELEMENT, SCALE FACTOR=*f*

Use the following option to scale the stable time increment based on the fixed time increment on the initial element-by-element estimate:

**DYNAMIC*, EXPLICIT, FIXED TIME INCREMENTATION, SCALE FACTOR=*f*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Time scaling factor**: *f*

Automatic Time Incrementation

The default time incrementation scheme in Abaqus/Explicit is fully automatic and requires no user intervention. Two types of estimates are used to determine the stability limit: element by element and global. An analysis always starts by using the element-by-element estimation method and might switch to the global estimation method under certain circumstances, as explained below.

Element-by-Element Estimation

In an analysis Abaqus/Explicit initially uses a stability limit based on the highest element frequency in the whole model. This element-by-element estimate is determined using the current dilatational wave speed in each element.

The element-by-element estimate is conservative; it will give a smaller stable time increment than the true stability limit that is based upon the maximum frequency of the entire model. In general, constraints such as boundary conditions and kinematic contact have the effect of compressing the eigenvalue spectrum, and the element-by-element estimates do not take this into account.

The concept of the stable time increment as the time required to propagate a dilatational wave across the smallest element dimension is useful for interpreting how the explicit procedure chooses the time increment when

element-by-element stability estimation controls the time increment. As the step proceeds, the global stability estimate, if used, makes the time increment less sensitive to element size.

Input File Usage: **DYNAMIC*, EXPLICIT, ELEMENT BY ELEMENT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Stable increment estimator**: **Element-by-element**

Global Estimation

The stability limit will be determined by the global estimator as the step proceeds unless the element-by-element estimation method is specified, fixed time incrementation is specified, or one of the conditions explained below prevents the use of global estimation. The switch to the global estimation method occurs once the algorithm determines that the accuracy of the global estimation method is acceptable.

The adaptive, global estimation algorithm determines the maximum frequency of the entire model using the current dilatational wave speed. This algorithm continuously updates the estimate for the maximum frequency. The global estimator will usually allow time increments that exceed the element-by-element values.

Abaqus/Explicit monitors the effectiveness of the global estimation algorithm. If the cost for computing the global time estimate is more than its benefit, the code will turn off the global estimation algorithm and simply use the element-by-element estimates to save computation time.

Conditions That Will Prevent the Use of the Global Time Estimator

The global estimation algorithm will not be used when any of the following capabilities are included in the model:

- Fluid elements
- Infinite elements
- Dashpots
- Thick shells (thickness to characteristic length ratio larger than 0.92)
- Thick beams (thickness to length ratio larger than 1.0)
- The JWL equation of state
- Material damping
- Nonisotropic elastic materials with temperature and field variable dependency
- Distortion control
- Adaptive meshing
- Subcycling

"Improved" Stable Time Increment for Three-Dimensional Continuum Elements and Elements with Plane Stress Formulations

For three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements) an "improved" estimate of the element characteristic length is used by default. This "improved" method usually results in a larger element stable time increment than a more traditional method. For analyses using variable mass scaling, the total mass added to achieve a given stable time increment will be less with the improved estimate.

Input File Usage: Use the following option to activate the "improved" element time estimation method:

*DYNAMIC, EXPLICIT, IMPROVED DT METHOD=YES

Use the following option to deactivate the “improved” element time estimation method:

*DYNAMIC, EXPLICIT, IMPROVED DT METHOD=NO

Abaqus/CAE Usage: Use the following option to activate the “improved” element time estimation method:

Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Automatic**: toggle on **Improved Dt Method**

Use the following option to deactivate the “improved” element time estimation method:

Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Automatic**: toggle off **Improved Dt Method**

Fixed Time Incrementation

A fixed time incrementation scheme is also available in Abaqus/Explicit. The fixed time increment size is determined either by the initial element-by-element stability estimate for the step or by a user-specified time increment.

Fixed time incrementation might be useful when a more accurate representation of the higher mode response of a problem is required. In this case a time increment size smaller than the element-by-element estimates might be used. The element-by-element estimate can be obtained simply by running a data check analysis (see *Abaqus/Standard and Abaqus/Explicit Execution*).

When fixed time incrementation is used, Abaqus/Explicit will not check that the computed response is stable during the step. You should ensure that a valid response has been obtained by carefully checking the energy history and other response variables.

Basing the Fixed Time Increment Size on the Initial Element-by-Element Stability Limit

You can use time increments the size of the initial element-by-element stability limit throughout a step. The dilatational wave speed in each element at the beginning of the step is used to compute the fixed time increment size.

Input File Usage: *DYNAMIC, EXPLICIT, FIXED TIME INCREMENTATION

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Fixed**: **Use element-by-element time increment estimator**

Specifying the Fixed Time Increment Size Directly

Alternatively, you can specify a time increment size directly.

Input File Usage: *DYNAMIC, EXPLICIT, DIRECT USER CONTROL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Fixed**: **User-defined time increment**

Advantages of the Explicit Method

The use of small increments (dictated by the stability limit) is advantageous because it allows the solution to proceed without iterations and without requiring tangent stiffness matrices to be formed. It also simplifies the treatment of contact.

The explicit dynamics procedure is ideally suited for analyzing high-speed dynamic events, but many of the advantages of the explicit procedure also apply to the analysis of slower (quasi-static) processes. A good example is sheet metal forming, where contact dominates the solution and local instabilities might form due to wrinkling of the sheet.

The results in an explicit dynamics analysis are not automatically checked for accuracy as they are in Abaqus/Standard (Abaqus/Standard uses the half-increment residual). In most cases this is not of concern because the stability condition imposes a small time increment such that the solution changes only slightly in any one time increment, which simplifies the incremental calculations. While the analysis might take an extremely large number of increments, each increment is relatively inexpensive, often resulting in an economical solution. It is not uncommon for Abaqus/Explicit to take over 10^5 increments for an analysis. The method is, therefore, computationally attractive for problems where the total dynamic response time that must be modeled is only a few orders of magnitude longer than the stability limit; for example, wave propagation studies or some "event and response" applications.

Computational Cost

The computer time involved in running a simulation using explicit time integration with a given mesh is proportional to the time period of the event. The time increment based on the element-by-element stability estimate can be rewritten (ignoring damping) in the form

$$\Delta t \leq \min\left(L_e \sqrt{\frac{\rho}{\widehat{\lambda}+2\widehat{\mu}}}\right),$$

where the minimum is taken over all elements in the mesh, L_e is a characteristic length associated with an element (see *Explicit dynamic analysis*), ρ is the density of the material in the element, and $\widehat{\lambda}$ and $\widehat{\mu}$ are the effective Lamé's constants for the material in the element (defined above).

The time increment from the global stability estimate might be somewhat larger, but for this discussion we will assume that the above inequality always holds (when the inequality does not hold, the solution time will be somewhat faster).

For linear, nonisotropic elastic materials this stability limit is further scaled down by the square root of the ratio of the effective material stiffness to the maximum material stiffness in one particular direction. Since this effectively means that the time increment can be no larger than the time required to propagate a stress wave across an element, the computer time involved in running a quasi-static analysis can be very large: the cost of the simulation is directly proportional to the number of time increments required.

The number of increments, n, required is $n = T/\Delta t$ if Δt remains constant, where T is the time period of the event being simulated. (Even the element-by-element approximation of Δt will not remain constant in general, since element distortion will change L_e and nonlinear material response will change the effective Lamé constants. But the assumption is sufficiently accurate for the purposes of this discussion.) Thus,

$$n \approx T \max\left(\frac{1}{L_e}\sqrt{\frac{\widehat{\lambda}+2\widehat{\mu}}{\rho}}\right).$$

In a two-dimensional analysis refining the mesh by a factor of two in each direction will increase the run time in the explicit procedure by a factor of eight—four times as many elements and half the original time increment size. Similarly, in a three-dimensional analysis refining the mesh by a factor of two in each direction will increase the run time by a factor of sixteen.

In a quasi-static analysis it is expedient to reduce the computational cost by either speeding up the simulation or by scaling the mass. In either case the kinetic energy should be monitored to ensure that the ratio of kinetic energy to internal energy does not get too large—typically less than 10%.

Reducing the Computational Cost by Speeding Up the Simulation

To reduce the number of increments required, n, we can speed up the simulation compared to the time of the actual process—that is, we can artificially reduce the time period of the event, T. This will introduce two possible errors. If the simulation speed is increased too much, the increased inertia forces will change the predicted response (in an extreme case the problem will exhibit wave propagation response). The only way to avoid this error is to choose a speed-up that is not too large.

The other error is that some aspects of the problem other than inertia forces—for example, material behavior—might also be rate dependent. In this case the actual time period of the event being modeled cannot be changed.

Reducing the Computational Cost by Using Mass Scaling

Artificially increasing the material density, ρ, by a factor f^2 reduces n to n/f, just like decreasing T to T/f. This concept, called "mass scaling," reduces the ratio of the event time to the time for wave propagation across an element while leaving the event time fixed, which allows rate-dependent behavior to be included in the analysis. Mass scaling has exactly the same effect on inertia forces as speeding up the time of simulation.

Mass scaling is attractive because it can be used in rate-dependent problems, but it must be used with care to ensure that the inertia forces do not dominate and change the solution. Either fixed or variable mass scaling can be invoked (see *Mass Scaling*).

Mass scaling can also be accomplished by altering the density; however, the fixed and variable mass scaling capabilities provide more versatile methods of scaling the mass of the entire model or specific element sets in the model.

Reducing the Computational Cost by Using Selective Subcycling

One disadvantage in an explicit dynamic analysis is that a few very small elements will force the entire model to be integrated with a small time increment. You can use mixed time integration or "subcycling" methods to reduce this problem. In these methods the equations of motion for the body are still integrated using the explicit central-difference integration rule as shown above, but the different time increments are allowed for different groups of nodes in the finite element model. If most nodes are integrated with a large stable time increment and only a few nodes are integrated with a small time increment, the computational cost might be reduced significantly.

Selective subcycling can be invoked by defining the subcycling zones. See *Selective Subcycling* for details.

Bulk Viscosity

Bulk viscosity introduces damping associated with volumetric straining. Its purpose is to improve the modeling of high-speed dynamic events (see *Stability* above for a discussion of the effect of damping on the stable time increment). Abaqus/Explicit contains two forms of bulk viscosity: linear and quadratic. Linear bulk viscosity is included by default in an Abaqus/Explicit analysis.

The bulk viscosity parameters b_1 and b_2 defined below can be redefined and can be changed from step to step. If the default values are changed in a step, the new values will be used in subsequent steps until they are redefined.

Bulk viscosities defined this way apply to the whole model. For an individual element set the linear and quadratic bulk viscosities can be scaled by a factor by defining section controls (see *Section Controls*).

Input File Usage: Use the following option to define bulk viscosity for the entire model:

**BULK VISCOSITY*

Use the following options to define bulk viscosity for an individual element set:

**BULK VISCOSITY*
**SECTION CONTROLS*

Abaqus/CAE Usage: Use the following option to define bulk viscosity for the entire model:

Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Other**: **Linear bulk viscosity parameter** and **Quadratic bulk viscosity parameter**

Defining bulk viscosity for an individual element set is not supported in Abaqus/CAE.

Linear Bulk Viscosity

Linear bulk viscosity is found in all elements and is introduced to damp "ringing" in the highest element frequency. This damping is sometimes referred to as truncation frequency damping. It generates a bulk viscosity pressure that is linear in the volumetric strain rate

$$p_{bv1} = b_1 \rho c_d L_e \dot{\epsilon}_{vol},$$

where b_1 is a damping coefficient (default=.06), ρ is the current material density, c_d is the current dilatational wave speed, L_e is an element characteristic length, and $\dot{\epsilon}_{vol}$ is the volumetric strain rate.

For acoustic elements, the bulk viscosity pressure can be obtained from the above equation by using the relationship of the fluid particle velocity and the pressure rate (see *Coupled acoustic-structural medium analysis*) as

$$p_{bv1} = -\frac{b_1 L_e}{c} \dot{p},$$

where $\dot{p}$ and c are the pressure rate and the speed of sound in the fluid, respectively.

Quadratic Bulk Viscosity

The second form of bulk viscosity pressure is found only in solid continuum elements (except the plane stress element CPS4R). This form is quadratic in the volumetric strain rate

$$p_{bv2} = \rho (b_2 L_e \dot{\epsilon}_{vol})^2,$$

where b_2 is a damping coefficient (default=1.2) and all other quantities are as defined for the linear bulk viscosity. Quadratic bulk viscosity is applied only if the volumetric strain rate is compressive.

The quadratic bulk viscosity pressure will smear a shock front across several elements and is introduced to prevent elements from collapsing under extremely high velocity gradients. Consider a simple one-element problem in which the nodes on one side of the element are fixed and the nodes on the other side have an initial velocity in the direction of the fixed nodes. If the initial velocity is equal to the dilatational wave speed of the material, without the quadratic bulk viscosity, the element would collapse to zero volume in one time increment (because the stable time increment size is precisely the transit time of a dilatational wave across the element). The quadratic bulk viscosity pressure will introduce a resisting pressure that will prevent the element from collapsing.

Fraction of Critical Damping due to Bulk Viscosity

The bulk viscosity pressure is not included in the material point stresses because it is intended as a numerical effect only—it is not considered part of the material's constitutive response. The bulk viscosity pressures are based upon the dilatational mode of each element. The fraction of critical damping in the dilatational mode of each element is given by

$$\xi = b_1 - b_2^2 \frac{L_e}{c_d} \min\left(0, \dot{\epsilon}_{vol}\right).$$

Rotational Bulk Viscosity for Shell Elements

For the displacement degrees of freedom, bulk viscosity introduces damping associated with volumetric straining. Linear bulk viscosity or truncation frequency damping is used to damp the high frequency ringing that leads to unwanted noise in the solution or spurious overshoot in the response amplitude. For the same reason, in shells the high frequency ringing in the rotational degrees of freedom is damped with linear bulk viscosity acting on the mean curvature strain rate. This damping generates a bulk viscosity "pressure moment," m, which is linear in the mean curvature strain rate

$$m = b_1 \frac{h_0^2}{12} \rho c_d L \dot{\kappa},$$

where b_1 is a damping coefficient (default = 0.06), h_0 is the original thickness, ρ is the mass density, c_d is the current dilatational wave speed, L is the characteristic length used for rotary inertia and transverse shear stiffness scaling (see *Finite-strain shell element formulation*), and $\dot{\kappa} = \dot{\kappa}_{11} + \dot{\kappa}_{22}$ is twice the mean curvature strain rate. The resultant pressure moment mh, where h is the current thickness, is added to the direct components of the moment resultant.

Material Damping

Defining inelastic material behavior, dashpots, etc. will introduce energy dissipation into a model. In addition to these mechanisms, general ("Rayleigh") material damping can be introduced (see *Material Damping*). Adding damping to a model, especially stiffness proportional damping, β_R, might significantly reduce the stable time increment.

Input File Usage: **DAMPING*, ALPHA=α_R, BETA=β_R

Abaqus/CAE Usage: Property module: material editor: **Mechanical**->**Damping**: **Alpha** and **Beta**

Obtaining Diagnostic Information about Critical Elements

Abaqus/Explicit writes critical elements (elements with the smallest stable time increments) and their stable time increment values to the output database at each summary increment for visualization in Abaqus/CAE. By default, the number of critical elements written to the output database is 10.

Input File Usage: **DIAGNOSTICS*, CRITICAL ELEMENTS=*value*

Abaqus/CAE Usage: The ability to control the number of critical elements written to the output database is not supported in Abaqus/CAE.

Obtaining Diagnostic Information about the Deformation Speed

The deformation speed in an element is defined as the largest absolute value of all the deformation rate components of an element times the element characteristic length, L_e. You can request diagnostic information about the deformation speed within a step definition, as described below. In a multistep analysis diagnostic requests remain in effect until they are explicitly redefined.

Deformation Speed Warnings

By default, Abaqus/Explicit will check for a relatively large deformation speed in all the elements since too high a value might cause the element to deform or collapse unrealistically. A warning message is issued if the ratio of deformation speed versus dilatational wave speed in an element reaches the value specified for the "warning ratio." By default, the warning ratio is 0.3. You can redefine this limit.

The first occurrence of the warning message is written to the status (`.sta`) file; subsequent occurrences are written to the message (`.msg`) file. See *About Output* for a description of these output files.

Generally when the ratio of deformation speed to dilatational wave speed is greater than 0.3, it is an indication that the purely mechanical material constitutive relationship is no longer valid and that a thermomechanical equation of state material is required.

Input File Usage: **DIAGNOSTICS*, WARNING RATIO=*ratio*

Abaqus/CAE Usage: The ability to redefine the warning ratio limit is not supported in Abaqus/CAE.

Deformation Speed Errors

An error message is issued and the analysis is terminated when the maximum ratio of deformation speed versus current dilatational wave speed for any element is greater than the "cutoff ratio." By default, the cutoff ratio is 1.0. You can redefine this limit.

The check for this cutoff ratio is not applied to any model that has an equation of state material (see *Equation of State*) or a user-defined material (see *User-Defined Mechanical Material Behavior*).

Input File Usage: **DIAGNOSTICS*, CUTOFF RATIO=*ratio*

Abaqus/CAE Usage: The ability to redefine the cutoff ratio limit is not supported in Abaqus/CAE.

Obtaining a Summary of the Deformation Speed Information

You can request summary diagnostic information to obtain warning and error messages for only the element with the largest ratio of deformation speed to dilatational wave speed.

Input File Usage: **DIAGNOSTICS*, DEFORMATION SPEED CHECK=SUMMARY

Abaqus/CAE Usage: A summary of the deformation speed diagnostic information is output by default in Abaqus/CAE.

Obtaining Detailed Deformation Speed Information

You can request detailed diagnostic information to obtain warning and error messages for all elements with large deformation speed to dilatational wave speed ratios.

Input File Usage: **DIAGNOSTICS*, DEFORMATION SPEED CHECK=DETAIL

Abaqus/CAE Usage: You cannot output detailed diagnostic information about the deformation speed in Abaqus/CAE.

Disabling Deformation Speed Checks

You can choose to completely bypass the checks for large deformation speed.

Input File Usage: **DIAGNOSTICS*, DEFORMATION SPEED CHECK=OFF

Abaqus/CAE Usage: You cannot disable the deformation speed checks in Abaqus/CAE.

Monitoring Output Variables for Extreme Values

There are some analyses in which it is useful to monitor the value of a variable at every increment. For example, in a force-driven analysis such as hydro-forming, the simulation time that is sufficient to model the completion of the physical process might depend on the magnitude of the displacement of a node or a group of nodes in the model. Another example is a drop test simulation where the postfailure response is not of interest. Monitoring the values of critical variables and halting the analysis when those variables exceed a given criterion can reduce computational expense and turnaround time.

For such problems Abaqus/Explicit allows output variables to be monitored during an analysis to verify whether or not their values have exceeded or fallen below user-specified values in specified element or node sets. Comparisons of specified element integration point variables, element section variables, or nodal variables with user-specified values are performed at every increment. At the first occurrence of a variable exceeding the user-specified bounds, the variable name, the associated element or node number, and the increment number are written to the status (`.sta`) file. In addition, you can request that the analysis be stopped and/or the output state be written in the increment following the one in which the variable has exceeded the user-specified bound. At the end of each step in which variables are monitored, the maximum, minimum, or absolute maximum value that each variable attains during the course of the analysis, along with the number of the element or node where the extreme value occurred, will be written to the status file.

Defining the Element and Nodal Variables to Be Monitored

The element output variables that can be monitored include all the element integration point variables and element section point variables that are available for history-type output to the output database. Similarly, the nodal output variables that can be monitored include all the nodal variables that are available for history output to the output database. The keys identifying the output variables are defined in *Abaqus/Explicit Output Variable Identifiers*.

Input File Usage: Use the first option with one or both of the following options in the history portion of the input file:

**EXTREME VALUE*
**EXTREME ELEMENT VALUE*, ELSET=*element_set_name*
**EXTREME NODE VALUE*, NSET=*nset_set_name*

The **EXTREME VALUE* option can be repeated in the same step, and the **EXTREME ELEMENT VALUE* and **EXTREME NODE VALUE* options can be repeated as many times as necessary.

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Halting the Analysis When the Extreme Value Criterion Is Met

You can choose to halt the analysis when the extreme value criterion is met. The analysis will stop at the end of the increment following the one in which any of the specified element or nodal variables exceeded the prescribed bounds.

Input File Usage: Use the following options:

**EXTREME VALUE*, HALT=YES
**EXTREME ELEMENT VALUE and/or *EXTREME NODE VALUE*

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Obtaining Output When the Extreme Value Criterion Is Met

You can obtain field-type output to the output database and an additional restart state when any of the selected variables fall outside the specified bounds for the first time during the analysis. The output will be written in the increment following the one in which such an occurrence took place. Since output is automatically written when the analysis terminates, this request has an effect only if you have not chosen to halt the analysis when the extreme value criterion is met as described above.

Input File Usage: Use either or both of the following options in conjunction with the **EXTREME VALUE* option:

**EXTREME ELEMENT VALUE*, ELSET=*element_set_name*, OUTPUT=YES
**EXTREME NODE VALUE*, NSET=*node_set_name*, OUTPUT=YES

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Monitoring Variables in a Multistep Analysis

In a multistep analysis the monitoring requests you specify remain in effect until they are redefined. You must redefine all requests to add or change any variables, element or node sets, or maxima or minima.

Stopping the Monitoring of Variables in a New Step

You can stop monitoring variables in a new step.

Input File Usage: Use the **EXTREME VALUE* option without the **EXTREME ELEMENT VALUE* and **EXTREME NODE VALUE* options.

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Controlling Material Rate Dependence

You can control whether to consider or ignore material rate dependence within the step in an explicit dynamic response. A step that ignores material rate dependence should not follow a step or analysis that considers material rate dependence.

The fully relaxed long-term elastic response is used for viscoelastic materials when the step ignores material rate dependence. The rate dependence of the material response defined via user subroutines *VUCREEPNETWORK*, *VUHARD*, and *VUMAT* can be ignored when the step ignores material rate dependence.

Input File Usage: Use the following option to consider material rate dependence within the step (default):

**DYNAMIC*, EXPLICIT, RATE DEPENDENCE=ON

Use the following option to ignore material rate dependence within the step:

**DYNAMIC*, EXPLICIT, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers material rate dependence.

Initial Conditions

Initial Conditions describes all of the initial conditions that are available for an explicit dynamic analysis.

Boundary Conditions

Boundary conditions can be defined as explained in *Boundary Conditions*. Boundary conditions applied during an explicit dynamic response step should use appropriate amplitude references (*Amplitude Curves*). If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference will be ignored, and a zero velocity boundary condition will be enforced.

Loads

The loading types available for an explicit dynamic analysis are explained in *About Loads*. Concentrated nodal forces or moments can be applied to the displacement or rotation degrees of freedom (1–6). Distributed pressure forces or body forces can also be applied; the distributed load types available with particular elements are described in *Abaqus Elements Guide*.

As with boundary conditions, loads applied during a dynamic response step should use appropriate amplitude references (*Amplitude Curves*). If loads are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step.

Predefined Fields

The following predefined fields can be specified, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in explicit dynamic analysis, you can specify nodal temperatures. Any difference between the applied and initial temperatures causes thermal strain if a thermal

expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.
- Although pore fluid pressure is not a degree of freedom in an explicit dynamic analysis, you can specify nodal pore fluid pressure as a predefined field variable (see *Pore Fluid Pressure*). The total stress is computed as the sum of the effective stress and the pore fluid pressure. It is assumed that the solid grains are incompressible (the effective strain is equal to the total strain).

Material Options

Any of the material models in Abaqus/Explicit can be used in a general explicit dynamic analysis (see *Combining Material Behaviors*).

Elements

All of the elements available in Abaqus/Explicit can be used in an explicit dynamic analysis. The elements are listed in *Abaqus Elements Guide*.

If coupled temperature-displacement elements are used in an explicit dynamic analysis, the temperature degrees of freedom will be ignored.

Output

The element output available for a dynamic analysis includes stress; strain; energies; and the values of state, field, and user-defined variables. The nodal output available includes displacements, velocities, accelerations, reaction forces, and coordinates. All the output variable identifiers are outlined in *Abaqus/Explicit Output Variable Identifiers*. The types of output available are described in *About Output*.

When an Abaqus/Explicit analysis encounters a fatal error, the preselected variables applicable to the current procedure are added automatically to the output database as field data for the last increment.

Energy output is particularly important in checking the accuracy of the solution in an explicit dynamic analysis. In general, the total energy (ETOTAL) should be a constant or close to a constant; the “artificial” energies, such as the artificial strain energy (ALLAE), the damping dissipation (ALLVD), and the mass scaling work (ALLMW) should be negligible compared to “real” energies such as the recoverable strain energy (ALLSE) and the kinetic energy (ALLKE).

In a quasi-static analysis the value of the kinetic energy (ALLKE) should not exceed a small fraction of the value of the total strain energy (ALLIE).

It is a good practice to output the constraint penalty work (ALLCW) and the contact penalty work (ALLPW) in analyses involving constraints (such as ties and fasteners) and contact. The value of these energies should be close to zero.

Input File Template

```
*HEADING
 …
*MATERIAL, NAME=name
*ELASTIC
 …
*DENSITY
Data lines to define density
```

```
*DAMPING, ALPHA=α, BETA=β
Data lines to define Rayleigh damping
 …
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=type
Data lines to specify initial conditions
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
*DIAGNOSTICS, DEFORMATION SPEED CHECK=SUMMARY
*BOUNDARY, AMPLITUDE=name
Data lines to describe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loading
*TEMPERATURE and/or *FIELD
Data lines to specify predefined fields
*FILE OUTPUT, NUMBER INTERVAL=n
*EL FILE
Data line specifying element output variables
*NODE FILE
Data line specifying node output variables
*ENERGY FILE
*OUTPUT, FIELD, NUMBER INTERVAL=n
*ELEMENT OUTPUT
Data line specifying element output variables
*NODE OUTPUT
Data line specifying node output variables
*OUTPUT, HISTORY, TIME INTERVAL=t
*ELEMENT OUTPUT, ELSET=element set name
Data line specifying element output variables
*NODE OUTPUT, NSET=node set name
Data line specifying node output variables
*ENERGY OUTPUT
Data line specifying energy output variables
*END STEP
**************************
*STEP
*DYNAMIC, EXPLICIT, ELEMENT BY ELEMENT
 …
*BULK VISCOSITY
Data line to define linear and/or quadratic bulk viscosity in this step
 …
*END STEP
```

Direct-Solution Steady-State Dynamic Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A direct-solution steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is a linear perturbation procedure;
- calculates the response directly in terms of the physical degrees of freedom of the model;
- is an alternative to mode-based steady-state dynamic analysis, in which the response of the system is calculated on the basis of the eigenmodes;
- is more expensive computationally than mode-based or subspace-based steady-state dynamics;
- is more accurate than mode-based or subspace-based steady-state dynamics, in particular if significant frequency-dependent material damping or viscoelastic material behavior is present in the structure; and
- is able to bias the excitation frequencies toward the approximate values that generate a response peak.

References:

- *About Dynamic Analysis Procedures*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*
- *Defining an Analysis*
- *General and Perturbation Procedures*
- *Output to the Output Database*
- **STEADY STATE DYNAMICS*
- *Configuring a direct-solution steady-state dynamic procedure*
- *Creating and modifying prescribed conditions*

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system due to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep by applying the loading at a series of different frequencies and recording the response; in Abaqus/Standard the direct-solution steady-state dynamic procedure conducts this frequency sweep. In a direct-solution steady-state analysis the steady-state harmonic response is calculated directly in terms of the physical degrees of freedom of the model using the mass, damping, and stiffness matrices of the system.

When defining a direct-solution steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below (*Selecting the Frequency Spacing*). Logarithmic frequency spacing is the default. Frequencies are given in cycles/time.

Those frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (described below).

The direct-solution steady-state analysis procedure can be used in the following cases for which the eigenvalues cannot be extracted (and, thus, the mode-based steady-state dynamics procedures are not applicable):

- for nonsymmetric stiffness;
- when any form of damping other than modal damping must be included; and
- when viscoelastic material properties must be taken into account.

While the response in this procedure is linear, the prior response can be nonlinear. Initial stress effects (stress stiffening) as well as load stiffness effects will be included in the steady-state dynamics response if nonlinear geometric effects (*General and Perturbation Procedures*) were included in any general analysis step prior to the direct-solution steady-state dynamic procedure.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**

Ignoring Damping

If damping terms can be ignored, you can specify that a real, rather than a complex, system matrix be factored, which can significantly reduce computational time. Damping is discussed below.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT, REAL ONLY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Compute real response only**

Selecting the Type of Frequency Interval for Which Output Is Requested

Three types of frequency intervals are permitted for output from a direct-solution steady-state dynamic step. If an eigenvalue extraction step precedes the direct-solution steady-state dynamic step, you can select either the range or the eigenfrequency type of frequency interval; otherwise, only the range type can be used.

Dividing the Specified Frequency Range Using the User-Defined Number of Points and the Optional Bias Function

For the range type of frequency interval (the default), the specified frequency range of interest is divided using the user-defined number of points and the optional bias function.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Specifying the Frequency Ranges by Using the System's Eigenfrequencies

If the direct-solution steady-state dynamic analysis is preceded by an eigenfrequency extraction step, you can select the eigenfrequency type of frequency interval. The following intervals then exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function. *Figure 1* illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

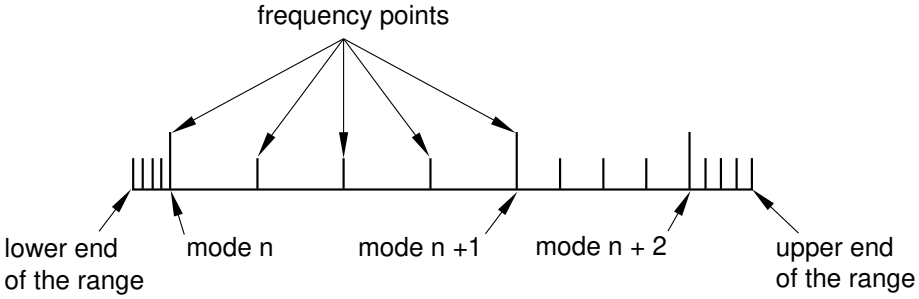


Figure 1: Division of range for the eigenfrequency type of interval and 5 calculation points.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Use eigenfrequencies to subdivide each frequency range**

Specifying the Frequency Ranges by the Frequency Spread

If the direct-solution steady-state dynamic analysis is preceded by an eigenfrequency extraction step, you can select the spread type of frequency interval. In this case intervals exist around each eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. *Figure 2* illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

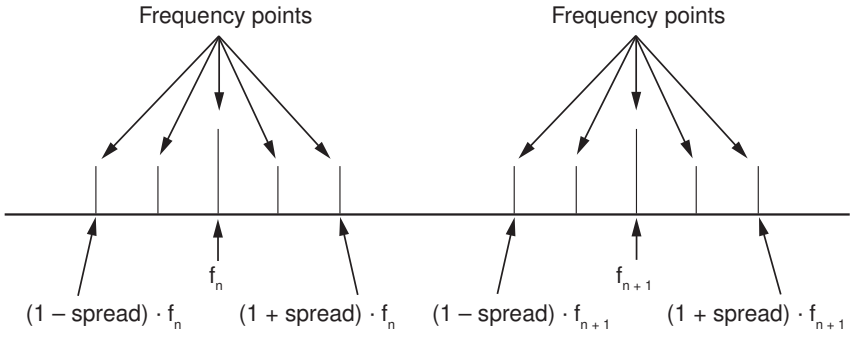


Figure 2: Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

Selecting the Frequency Spacing

Two types of frequency spacing are permitted for a direct-solution steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use either of the following options:

**STEADY STATE DYNAMICS*, DIRECT,
FREQUENCY SCALE=LOGARITHMIC
**STEADY STATE DYNAMICS*, DIRECT, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Scale: Logarithmic** or **Linear**

Requesting Multiple Frequency Ranges

You can request multiple frequency ranges or multiple single frequency points for a direct-solution steady-state dynamic step.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...

Repeat the data lines as often as necessary.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Data**: enter data in table, and add rows as necessary

The Bias Parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. *Figure 3* shows a few examples of the effect of the bias parameter on the frequency spacing.

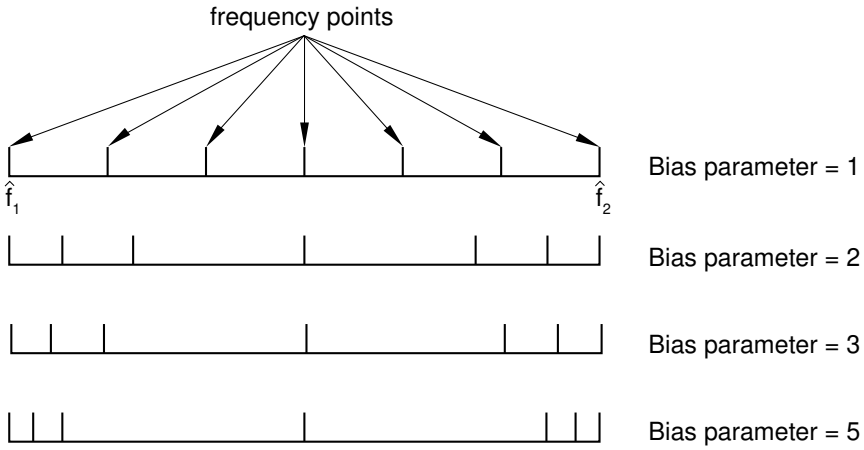


Figure 3: Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used in direct-solution steady-state dynamics is

$$\hat{f}_k = \tfrac{1}{2}\left(\hat{f}_1 + \hat{f}_2\right) + \tfrac{1}{2}\left(\hat{f}_2 - \hat{f}_1\right)|y|^{1/p}\,\mathrm{sign}\,(y)\,,$$

where

y

$= -1 + 2\,(k-1)\,/\,(n-1)$;

n

is the number of frequency points at which results are to be given;

k

is one such frequency point ($k = 1, 2, \ldots, n$);

$\hat{f}_1$

is the lower limit of the frequency range;

$\hat{f}_2$

is the upper limit of the range;

$\hat{f}_k$

is the frequency at which the *k*th results are given;

p

is the bias parameter value; and

$\hat{f}$

is the frequency or the logarithm of the frequency, depending on the value chosen for the frequency scale.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 1.0 for a range frequency interval and 3.0 for an eigenfrequency interval.

The Frequency Scale Factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see *Specifying the Frequency Ranges by Using the System's Eigenfrequencies* above).

Damping

If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in *Material Damping*.

In direct-solution steady-state dynamics damping can be created by the following:

- dashpots (see *Dashpots*),
- “Rayleigh” damping associated with materials and elements (see *Material Damping*),
- damping associated with acoustic elements (see *Acoustic Medium*, *Infinite Elements*, and *Acoustic and Shock Loads*),
- structural damping (see *Damping in Dynamic Analysis*), and
- viscoelasticity included in the material definitions (see *Frequency Domain Viscoelasticity*).

When a real-only system matrix is factored, all forms of damping are ignored, including quiet boundaries on infinite elements and nonreflecting boundaries on acoustic elements.

Contact Conditions with Sliding Friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom are not constrained and the effect of friction results in an unsymmetric contribution to the stiffness matrix. At other contact nodes the tangential degrees of freedom are constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing and is also known as “negative damping.” For more details, see *Coulomb friction*. Direct-solution steady-state dynamics analysis allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: **STEADY STATE DYNAMICS*, DIRECT, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Include friction-induced damping effects**

Initial Conditions

The base state is the current state of the model at the end of the last general analysis step prior to the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions (*Initial Conditions*). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary Conditions

In a steady-state dynamic analysis the real and imaginary parts of any degree of freedom are either restrained or unrestrained simultaneously; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will automatically restrain both the real and imaginary parts of a degree of freedom even if only one part is prescribed specifically. The unspecified part will be assumed to have a perturbation magnitude of zero.

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) in a direct-solution steady-state analysis. See *Boundary Conditions*. These boundary conditions will vary sinusoidally with time. You specify the real (in-phase) part of a boundary condition and the imaginary (out-of-phase) part of a boundary condition separately.

Input File Usage: Use either of the following options to define the real (in-phase) part of the boundary condition:

**BOUNDARY*
**BOUNDARY*, REAL

Use the following option to define the imaginary (out-of-phase) part of the boundary condition:

**BOUNDARY*, IMAGINARY

Abaqus/CAE Usage: Load module: boundary condition editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**

Frequency-Dependent Boundary Conditions

An amplitude definition can be used to specify the amplitude of a boundary condition as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**BOUNDARY*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:***name*

Load module: boundary condition editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**: **Amplitude:***name*

Loads

The following loads can be prescribed in a steady-state dynamic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.
- Incident wave loads can be applied; see *Acoustic and Shock Loads*.

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Coriolis distributed loading adds an imaginary antisymmetric contribution to the overall system of equations. This contribution is currently accounted for in solid and truss elements only and is activated by using the unsymmetric matrix storage and solution scheme for the step (*Defining an Analysis*).

Incident wave loads can be used to model sound waves from distinct planar or spherical sources or from diffuse fields.

Fluid flux loading cannot be used in a steady-state dynamic analysis.

Input File Usage: Use any of the following options to define the real (in-phase) part of the load:

**CLOAD or *DLOAD*
**CLOAD or *DLOAD*, REAL

Use either of the following options to define the imaginary (out-of-phase) part of the load:

**CLOAD or *DLOAD*, IMAGINARY

Abaqus/CAE Usage: Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**

Frequency-Dependent Loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**CLOAD or *DLOAD*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:***name*

Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**: **Amplitude:***name*

Predefined Fields

Predefined temperature fields can be specified in direct-solution steady-state dynamic analysis (see *Predefined Fields*) and can produce harmonically varying thermal strains if thermal expansion is included in the material definition (see *Computing Thermal Strains in Linear Perturbation Steps*). Other predefined fields are ignored.

Material Options

As in any dynamic analysis procedure, mass or density (*Density*) must be assigned to some regions of any separate parts of the model where dynamic response is required. If an analysis is desired in which the inertia effects are neglected, the density should be set to a very small number. The following material properties are not active during steady-state dynamic analyses: plasticity and other inelastic effects, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see *General and Perturbation Procedures*.

Viscoelastic effects can be included in direct-solution steady-state harmonic response analysis. The linearized viscoelastic response is considered to be a perturbation about a nonlinear preloaded state, which is computed on the basis of purely elastic behavior (long-term response) in the viscoelastic components. Therefore, the vibration amplitude must be sufficiently small so that the material response in the dynamic phase of the problem can be treated as a linear perturbation about the predeformed state. Viscoelastic frequency domain response is described in *Frequency Domain Viscoelasticity*.

Elements

Any of the following elements available in Abaqus/Standard can be used in a steady-state dynamic procedure:

- stress/displacement elements (other than generalized axisymmetric elements with twist);
- acoustic elements;
- piezoelectric elements; or
- hydrostatic fluid elements.

See *Choosing the Appropriate Element for an Analysis Type*.

Output

In direct-solution steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see *Abaqus/Standard Output Variable Identifiers*). In the case of data file output the first printed line gives the magnitudes while the second lists the phase angle.

In steady-state dynamic analysis procedures, you can request output for load cases to store only relevant results for each load case. This can reduce the size of the output database.

The following variables are provided specifically for steady-state dynamic analysis:

Element integration point variables:

PHS Magnitude and phase angle of all stress components.

PHE Magnitude and phase angle of all strain components.

PHEPG Magnitude and phase angles of the electrical potential gradient vector.

PHEFL Magnitude and phase angles of the electrical flux vector.

PHMFL Magnitude and phase angle of the mass flow rate in fluid link elements.

PHMFT Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF Magnitude and phase angle of connector total forces.

PHCEF Magnitude and phase angle of connector elastic forces.

PHCVF Magnitude and phase angle of connector viscous forces.

PHCRF Magnitude and phase angle of connector reaction forces.

PHCSF Magnitude and phase angle of connector friction forces.

PHCU Magnitude and phase angle of connector relative displacements.

PHCCU Magnitude and phase angle of connector constitutive displacements.

PHCV Magnitude and phase angle of connector relative velocities.

PHCA Magnitude and phase angle of connector relative accelerations.

Nodal variables:

PU Magnitude and phase angle of all displacement/rotation components at a node.

PPOR Magnitude and phase angle of the fluid, pore, or acoustic pressure at a node.

PHPOT Magnitude and phase angle of the electrical potential at a node.

PRF Magnitude and phase angle of all reaction forces/moments at a node.

PHCHG Magnitude and phase angle of the reactive charge at a node.

The elastic strain energy density (SENER) is not available for output in a SIM-based steady-state dynamic analysis.

Whole model variables such as ALLIE (total strain energy) are available for direct-solution steady-state dynamic analysis by requesting energy output to the data, results, or output database files (see *Output to the Data and Results Files* and *Output to the Output Database*).

Input File Template

```
*HEADING
 …
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamic step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
…
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, DIRECT
Data lines to specify frequency ranges and bias parameters
*BOUNDARY, REAL
Data lines to specify real (in-phase) boundary conditions
*BOUNDARY, IMAGINARY
Data lines to specify imaginary (out-of-phase) boundary conditions
*CLOAD, AMPLITUDE=loadamp
Data lines to specify sinusoidally varying, frequency-dependent, concentrated loads
*CLOAD and/or *DLOAD
Data lines to specify sinusoidally varying loads
…
*END STEP
```

Natural Frequency Extraction

Products: Abaqus/Standard Abaqus/CAE Abaqus/AMS

Overview

The frequency extraction procedure:

- performs eigenvalue extraction to calculate the natural frequencies and the corresponding mode shapes of a system;
- includes initial stress and load stiffness effects due to preloads and initial conditions if geometric nonlinearity is accounted for in the base state, so that small vibrations of a preloaded structure can be modeled;
- computes residual modes if requested;
- is a linear perturbation procedure;
- is performed using the traditional architecture if appropriate, but the high-performance SIM architecture (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*) is used as the default; and
- solves the eigenfrequency problem only for symmetric mass and stiffness matrices; the complex eigenfrequency solver must be used if unsymmetric contributions, such as the load stiffness, are required.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Dynamic Analysis Procedures*
- **FREQUENCY*
- *Configuring a frequency procedure*
- *Output to the Output Database*

Eigenvalue Extraction

The eigenvalue problem for the natural frequencies of an undamped finite element model is

$$\left(-\omega^2 M^{MN} + K^{MN}\right) \phi^N = 0,$$

where

M^{MN}

is the mass matrix (which is symmetric and positive definite);

K^{MN}

is the stiffness matrix (which includes initial stiffness effects if the base state included the effects of nonlinear geometry);

ϕ^N

is the eigenvector (the mode of vibration); and

M and N

are degrees of freedom.

When $\boldsymbol{K}^{MN}$ is positive definite, all eigenvalues are positive. Rigid body modes and instabilities cause $\boldsymbol{K}^{MN}$ to be indefinite. Rigid body modes produce zero eigenvalues. Instabilities produce negative eigenvalues and occur when you include initial stress effects. Abaqus/Standard solves the eigenfrequency problem only for symmetric matrices.

Selecting the Eigenvalue Extraction Method

Abaqus/Standard provides three eigenvalue extraction methods:

- Lanczos
- Automatic multilevel substructuring (AMS)
- Subspace iteration

In addition, you must consider the software architecture used for the subsequent modal superposition procedures. The choice of architecture has minimal impact on the frequency extraction procedure, but the default SIM architecture offers significant performance improvements over the traditional architecture for subsequent mode-based steady-state or transient dynamic procedures (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*). The architecture that you use for the frequency extraction procedure is used for all subsequent mode-based linear dynamic procedures; you cannot switch architectures during an analysis. For information about the software architectures used by the different eigensolvers, see *Table 1*.

Table 1: Software architectures available with different eigensolvers.

Software Architecture	Eigensolver		
	Lanczos	AMS	Subspace Iteration
Traditional	√		√
SIM	√	√	√

The Lanczos solver with the SIM architecture is the default eigenvalue extraction method because it has the most general capabilities. However, the Lanczos method is generally slower than the AMS method. The increased speed of the AMS eigensolver is particularly evident when you require a large number of eigenmodes for a system with many degrees of freedom. However, the AMS method has the following limitations:

- All restrictions imposed on SIM-based linear dynamic procedures also apply to mode-based linear dynamic analyses based on mode shapes computed by the AMS eigensolver. See *Using the SIM Architecture for Modal Superposition Dynamic Analyses* for details.
- The AMS eigensolver does not compute composite modal damping factors when selective recovery is requested.
- You cannot use the AMS eigensolver in an analysis that contains piezoelectric elements.
- You cannot request output to the results (`.fil`) file in an AMS frequency extraction step.
- The SIM-based architecture does not support the following capabilities: cyclic symmetry, fluid properties associated with an incident wave, transient dynamic procedures used for coupled structural-acoustic problems, symmetric model generation, and imperfections based on eigenmode data.
- At least one output request is required to run analyses using the SIM-based architecture.

If your model has many degrees of freedom and these limitations are acceptable, you can use the AMS eigensolver. Otherwise, you can use the Lanczos eigensolver. For a description of the Lanczos eigensolver and the subspace iteration method, see *Eigenvalue extraction*.

Lanczos Eigensolver

For the Lanczos method, you must provide the maximum frequency of interest or the number of eigenvalues required; Abaqus/Standard determines a suitable block size (although you can override this choice, if required). If you specify both the maximum frequency of interest and the number of eigenvalues required and the actual number of eigenvalues is underestimated, Abaqus/Standard issues a corresponding warning message.

You can also specify the minimum frequencies of interest; Abaqus/Standard extracts eigenvalues until either it extracts the requested number of eigenvalues in the given range or it extracts all the frequencies in the given range.

See *Using the SIM Architecture for Modal Superposition Dynamic Analyses* for information on using the SIM architecture with the Lanczos eigensolver.

Input File Usage: **FREQUENCY*, EIGENSOLVER=LANCZOS

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver**: **Lanczos**

Choosing a Block Size for the Lanczos Method

In general, the block size for the Lanczos method can be as large as the largest expected multiplicity of eigenvalues (that is, the largest number of modes with the same frequency). A block size larger than 10 is not recommended. If the number of eigenvalues requested is n, the default block size is the minimum of (7, n). The choice of 7 for block size proves to be efficient for problems with rigid body modes. Abaqus/Standard typically determines the number of block Lanczos steps within each Lanczos run, but you can change the number. In general, if a particular type of eigenproblem converges slowly, providing more block Lanczos steps reduces the analysis cost. However, if you know that a particular type of problem converges quickly, providing fewer block Lanczos steps reduces the amount of in-core memory used. The default values are

Block size	Maximum number of block Lanczos steps
1	80
2	50
3	45
4	35

Automatic Multilevel Substructuring (AMS) Eigensolver

For the AMS method, you only need to specify the maximum frequency of interest (the global frequency), and Abaqus/Standard extracts all the modes up to this frequency. You can also specify the minimum frequencies of interest and the number of requested modes. However, specifying these values does not affect the number of modes extracted by the eigensolver; it affects only the number of modes that Abaqus/Standard stores for output or for a subsequent modal analysis.

You can control the execution of the AMS eigensolver by specifying three parameters: AMS_{cutoff_1}, AMS_{cutoff_2}, and AMS_{cutoff_3}. These three parameters multiplied by the maximum frequency of interest define three cutoff frequencies. AMS_{cutoff_1} (default value of 5) controls the cutoff frequency for substructure eigenproblems in the

reduction phase, while AMS_{cutoff_2} and AMS_{cutoff_3} (default values of 1.7 and 1.1, respectively) control the cutoff frequencies used to define a starting subspace in the reduced eigensolution phase. Generally, increasing the value of AMS_{cutoff_2} and AMS_{cutoff_3} improves the accuracy of the results but might affect the performance of the analysis.

Requesting Eigenvectors at All Nodes

By default, the AMS eigensolver computes eigenvectors at every node of the model.

Input File Usage: **FREQUENCY*, EIGENSOLVER=AMS

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver**: **AMS**

Requesting Eigenvectors Only at Specified Nodes

Alternatively, you can specify a node set. Abaqus/Standard computes partial eigenvectors and stores them only at the nodes that belong to the specified node set. Computing eigenvectors at only selected nodes improves performance and reduces the amount of stored data. Therefore, it is recommended that you use this option for large models.

The node set that you specify must include all nodes at which loads are applied and the nodes at which output is requested in the current natural frequency extraction analysis or in any subsequent modal analysis, including any restarted analysis. If element output is requested or element-based loading is applied, you must also include all nodes attached to the associated elements in this node set, including the internal nodes created by Abaqus/Standard. You cannot specify internal nodes to include in the node set. However, Abaqus/Standard automatically selects all the nodes that need to be included in the node set. This guarantees that all nodes required for the load application and output specified in the current analysis job are included in the node set. If loads are applied or output is requested at nodes not included in this node set in a restart analysis, Abaqus/Standard issues diagnostic messages and stops the restarted analysis. The following nodes are selected automatically:

- nodes at which a concentrated load is applied in the mode-based procedures that follow this step,
- nodes at which output is requested in the eigenvalue extraction analysis or in the mode-based procedures that follow this step,
- nodes at which residual vectors are requested,
- nodes of elements at which a distributed load is applied,
- nodes of elements with frequency-dependent material properties, and
- nodes of elements at which output is requested in the eigenvalue extraction analysis or in the mode-based procedures that follow this step.

Input File Usage: Use the following option to specify a node set:

**FREQUENCY*, EIGENSOLVER=AMS, NSET=*name*

Use the following option to allow Abaqus/Standard to select the nodes automatically:

**FREQUENCY*, EIGENSOLVER=AMS, NSET

Abaqus/CAE Usage: You can only request eigenvectors at specific nodes by specifying a node set in Abaqus/CAE.

Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver**: **AMS**: **Limit region of saved eigenvectors**, select node set

Controlling the AMS Eigensolver

The AMS method consists of the following three phases:

Reduction phase

In this phase Abaqus/Standard uses a multilevel substructuring technique to reduce the full system in a way that allows a very efficient eigensolution of the reduced system. The approach combines a sparse factorization based on a multilevel supernode elimination tree and a local eigensolution at each supernode.

Starting from the lowest level supernodes, we use a Craig-Bampton substructure reduction technique to successively reduce the size of the system as we progress upward in the elimination tree. At each supernode a local eigensolution is obtained based on fixing the degrees of freedom connected to the next higher level supernode (these are the local retained or "fixed-interface" degrees of freedom). At the end of the reduction phase, the full system has been reduced such that the reduced stiffness matrix is diagonal and the reduced mass matrix has unit diagonal values but contains off-diagonal blocks of nonzero values representing the coupling between the supernodes.

The cost of the reduction phase depends on the system size and the number of eigenvalues extracted (the number of eigenvalues extracted is controlled indirectly by specifying the highest eigenfrequency required).

You can make trade-offs between cost and accuracy during the reduction phase through the AMS_{cutoff_1} parameter. This parameter multiplied by the highest eigenfrequency specified for the full model yields the highest eigenfrequency that is extracted in the local supernode eigensolution. Increasing the value of

AMS_{cutoff_1} increases the accuracy of the reduction since more local eigenmodes are retained. However, increasing the number of retained modes also increases the cost of the reduced eigensolution phase, which is discussed next.

Reduced eigensolution phase

In this phase Abaqus/Standard computes the eigensolution of the reduced system that comes from the previous phase. Although the reduced system typically is two orders of magnitude smaller in size than the original system, generally it still is too large to solve directly. Thus, the system is further reduced mainly by truncating the retained eigenmodes and then solved using a single subspace iteration step. The two

AMS parameters, AMS_{cutoff_2} and AMS_{cutoff_3}, define a starting subspace of the subspace iteration step. The default values of these parameters are carefully chosen and provide accurate results in most cases. When a more accurate solution is required, the recommended procedure is to increase both parameters proportionally from their respective default values.

Recovery phase

In this phase the eigenvectors of the original system are recovered using eigenvectors of the reduced problem and local substructure modes. If you request recovery at specified nodes, the eigenvectors are computed only at those nodes.

Subspace Iteration Eigensolver

For the subspace iteration eigensolver, you only need to specify the number of eigenvalues required; Abaqus/Standard chooses a suitable number of vectors for the iteration. If the subspace iteration technique is requested, you can also specify the maximum frequency of interest; Abaqus/Standard extracts eigenvalues until

either the requested number of eigenvalues has been extracted or the last frequency extracted exceeds the maximum frequency of interest.

Input File Usage: **FREQUENCY*, EIGENSOLVER=SUBSPACE

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver**: **Subspace**

Structural-Acoustic Coupling

Structural-acoustic coupling affects the natural frequency response of systems. All three eigensolvers in Abaqus/Standard can extract coupled structural-acoustic eigenmodes to fully include this effect. Alternatively, all eigensolvers can also compute uncoupled structural and acoustic eigenmodes. In this case, Abaqus/Standard computes the modes and frequencies using natural boundary conditions at the structural-acoustic coupling surface in the structural and acoustic domains independently. It projects the coupling onto the modal space and stores it for later use.

The method used by default depends on the eigensolver. For the Lanczos and subspace iteration eigensolvers, Abaqus/Standard extracts coupled structural-acoustic eigenmodes by default. For the AMS eigensolver, Abaqus/Standard computes uncoupled structural and acoustic eigenmodes by default.

Structural-Acoustic Coupling Using the Lanczos and Subspace Iteration Eigensolvers

If structural-acoustic coupling is present in the model and the Lanczos or subspace iteration methods are used, Abaqus/Standard extracts the coupled modes by default. Because these modes fully account for coupling, they represent the mathematically optimal basis for subsequent modal procedures. The effect is most noticeable in strongly coupled systems such as steel shells and water. However, coupled structural-acoustic modes cannot be used in subsequent random response or response spectrum analyses. You can define the coupling using either acoustic-structural interaction elements (see *Acoustic Interface Elements*) or surface-based tie constraints (see *Coupled Acoustic-Structural Analysis*). You can ignore coupling when extracting acoustic and structural modes; in this case Abaqus/Standard treats the coupling boundary as traction-free on the structural side and rigid on the acoustic side.

For frequency extractions that use the Lanczos or the subspace iteration eigensolver based on the SIM architecture, you can project the structural-acoustic coupling operators onto the subspace of eigenvectors. Abaqus/Standard computes the modes using traction-free boundary conditions on the structural side of the coupling boundary and rigid boundary conditions on the acoustic side. Structural-acoustic coupling operators (see *Coupled Acoustic-Structural Analysis*) are projected by default onto the subspace of eigenvectors. Contributions to these global operators, which come from surface-based tie constraints defined between the structural and acoustic surfaces, are assembled into global matrices that are projected onto the mode shapes and used in subsequent SIM-based modal dynamic procedures.

Input File Usage: Use the following option to account for structural-acoustic coupling during the frequency extraction:

**FREQUENCY*, EIGENSOLVER=LANCZOS|SUBSPACE,
ACOUSTIC COUPLING=ON (default, except when the subspace eigensolver is not based on the SIM architecture)

Use the following option to project structural-acoustic coupling on the uncoupled eigenmodes during the frequency extraction:

**FREQUENCY*, EIGENSOLVER=LANCZOS|SUBSPACE, SIM,
ACOUSTIC COUPLING=PROJECTION (only if the Lanczos and subspace eigensolvers are based on the SIM architecture)

Abaqus/CAE Usage: Use the following option to account for structural-acoustic coupling during the frequency extraction:

Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver: Lanczos | Subspace**, toggle on **Include acoustic-structural coupling where applicable**

Use the following option to project structural-acoustic coupling on the uncoupled eigenmodes during the frequency extraction:

Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver: Lanczos | Subspace**, toggle on **Use SIM-based linear dynamics procedures**, toggle on **Project acoustic-structural coupling where applicable**

Structural-Acoustic Coupling Using the AMS Eigensolver

By default, for frequency extractions that use the AMS eigensolver, Abaqus/Standard computes the modes using traction-free boundary conditions on the structural side of the coupling boundary and rigid boundary conditions on the acoustic side. Structural-acoustic coupling operators (see *Coupled Acoustic-Structural Analysis*) are projected by default onto the subspace of eigenvectors. Contributions to these global operators, which come from surface-based tie constraints defined between structural and acoustic surfaces, are assembled into global matrices that are projected onto the mode shapes and used in subsequent SIM-based modal dynamic procedures.

For frequency extractions that use the AMS eigensolver, Abaqus/Standard can also extract the coupled modes. Because these modes fully account for coupling, they represent the mathematically optimal basis for subsequent modal procedures. The effect is most noticeable in strongly coupled systems such as steel shells and water. However, extracting the coupled structural-acoustic modes using the AMS eigensolver is computationally more expensive than the default option, where the coupling operators are projected onto the subspace of uncoupled eigenvectors.

User-defined acoustic-structural interaction elements (see *Acoustic Interface Elements*) cannot be used in an AMS eigenvalue extraction analysis.

Input File Usage: Use the following option to project structural-acoustic coupling operators onto the subspace of eigenvectors:

**FREQUENCY*, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=PROJECTION (default)

Use the following option to disable the projection of structural-acoustic coupling operators:

**FREQUENCY*, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=OFF

Use the following option to extract coupled structural-acoustic eigenmodes:

**FREQUENCY*, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=ON

Abaqus/CAE Usage: Use the following option to project structural-acoustic coupling operators onto the subspace of eigenvectors:

Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, toggle on **Project acoustic-structural coupling where applicable**

Use the following option to disable the projection of structural-acoustic coupling operators:

Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, toggle off **Project acoustic-structural coupling where applicable**

Requesting the coupled structural-acoustic modes during the AMS eigenanalysis is not supported in Abaqus/CAE.

Specifying a Frequency Range for the Acoustic Modes

Because structural-acoustic coupling can be ignored during the AMS and SIM-based Lanczos and subspace iteration eigenanalysis, the computed resonances are, in principle, higher than those of the fully coupled system. This might be understood as a result of neglecting the mass of the fluid in the structural phase and vice versa. For the common metal and air case, the structural resonances might be relatively unaffected; however, some acoustic modes that are significant in the coupled response might be omitted due to the air's upward frequency shift during eigenanalysis. Therefore, Abaqus/Standard allows you to specify a multiplier, so that the maximum acoustic frequency in the analysis is taken to be higher than the structural maximum.

Input File Usage: Use one of the following options:

**FREQUENCY*, EIGENSOLVER=AMS
, , , , , , *acoustic range factor*

or

**FREQUENCY*, EIGENSOLVER=LANCZOS, SIM,
ACOUSTIC COUPLING=PROJECTION
, , , , , , *acoustic range factor*

or

**FREQUENCY*, EIGENSOLVER=SUBSPACE, SIM,
ACOUSTIC COUPLING=PROJECTION
, , , , , *acoustic range factor*

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, **Acoustic range factor**: *acoustic range factor*

Specifying a frequency range for the acoustic modes during Lanczos or subspace iteration eigenanalysis is not supported in Abaqus/CAE.

Effects of Fluid Motion on Natural Frequency Analysis of Acoustic Systems

To extract natural frequencies from an acoustic-only or coupled structural-acoustic system in which fluid motion is prescribed using an acoustic flow velocity, either the Lanczos method or the complex eigenvalue extraction procedure can be used. In the former case Abaqus/Standard extracts real-only eigenvalues and considers the fluid motion's effects only on the acoustic stiffness matrix. Thus, these results are of primary interest as a basis for subsequent linear perturbation procedures. When the complex eigenvalue extraction procedure is used, the fluid motion effects are included in their entirety; that is, the acoustic stiffness and damping matrices are included in the analysis.

Frequency Shift

For the Lanczos and subspace iteration eigensolvers you can specify a positive or negative shifted squared frequency, *S*. This feature is useful when a particular frequency is of concern or when the natural frequencies of an unrestrained structure or a structure that uses secondary base motions (large mass approach) are required. In the latter case a shift from zero (the frequency of the rigid body modes) avoids singularity problems or round-off errors for the large mass approach; a negative frequency shift is normally used. The default is no shift.

If the Lanczos eigensolver is in use and the user-specified shift is outside the requested frequency range, the shift is adjusted automatically to a value close to the requested range.

Normalization

For the Lanczos and subspace iteration eigensolvers both displacement and mass eigenvector normalization are available. If the non-SIM architecture is used, displacement normalization is the default. Mass normalization is the only option available for SIM-based natural frequency extraction.

The choice of eigenvector normalization type has no influence on the results of subsequent modal dynamic steps (see *Linear analysis of a rod under dynamic loading*). The normalization type determines only the manner in which the eigenvectors are represented.

In addition to extracting the natural frequencies and mode shapes, the Lanczos and subspace iteration eigensolvers automatically calculate the generalized mass, the participation factor, the effective mass, and the composite modal damping for each mode; therefore, these variables are available for use in subsequent linear dynamic analyses. The AMS eigensolver computes the generalized mass and the participation factor.

Displacement Normalization

If displacement normalization is selected, the eigenvectors are normalized so that the largest displacement entry in each vector is unity. If the displacements are negligible, as in a torsional mode, the eigenvectors are normalized so that the largest rotation entry in each vector is unity. In a coupled acoustic-structural extraction, if the displacements and rotations in a particular eigenvector are small when compared to the acoustic pressures, the eigenvector is normalized so that the largest acoustic pressure in the eigenvector is unity. The normalization is done before the recovery of dependent degrees of freedom that have been previously eliminated with multi-point constraints or equation constraints. Therefore, it is possible that such degrees of freedom might have values greater than unity.

Input File Usage: **FREQUENCY*, NORMALIZATION=DISPLACEMENT, SIM=NO

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Other**: **Normalize eigenvectors by**: **Displacement**

Mass Normalization

Alternatively, the eigenvectors can be normalized so that the generalized mass for each vector is unity.

The “generalized mass” associated with mode α is

$$m_\alpha = \phi_\alpha^N M^{NM} \phi_\alpha^M \text{ (no sum on } \alpha),$$

where M^{NM} is the structure's mass matrix and ϕ_α^N is the eigenvector for mode α. The superscripts *N* and *M* refer to degrees of freedom of the finite element model.

If the eigenvectors are normalized with respect to mass, all the eigenvectors are scaled so that m_α=1. For coupled acoustic-structural analyses, an acoustic contribution fraction to the generalized mass is computed as well.

Input File Usage: **FREQUENCY*, NORMALIZATION=MASS

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Other**: **Normalize eigenvectors by**: **Mass**

Modal Participation Factors

The participation factor for mode α in direction i, $\Gamma_{\alpha i}$, is a variable that indicates how strongly motion in the global x-, y-, or z-direction or rigid body rotation about one of these axes is represented in the eigenvector of that mode. The six possible rigid body motions are indicated by $i = 1, 2, \ldots, 6$. The participation factor is defined as

$$\Gamma_{\alpha i} = \frac{1}{m_\alpha} \phi_\alpha^N M^{NM} T_i^M \text{ (no sum on } \alpha),$$

where T_i^N defines the magnitude of the rigid body response of degree of freedom N in the model to imposed rigid body motion (displacement or infinitesimal rotation) of type i. For example, at a node with three displacement and three rotation components, T_i^N is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & (z-z_0) & -(y-y_0) \\ 0 & 1 & 0 & -(z-z_0) & 0 & (x-x_0) \\ 0 & 0 & 1 & (y-y_0) & -(x-x_0) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \\ \hat{e}_4 \\ \hat{e}_5 \\ \hat{e}_6 \end{Bmatrix},$$

where $\hat{e}_i$ is unity and all other $\hat{e}_p$ are zero; x, y, and z are the coordinates of the node; and x_0, y_0, and z_0 represent the coordinates of the center of rotation. The participation factors are, thus, defined for the translational degrees of freedom and for rotation around the center of rotation. For coupled acoustic-structural eigenfrequency analysis, an additional acoustic participation factor is computed as outlined in *Coupled acoustic-structural medium analysis*.

Modal Effective Mass

The effective mass for mode α associated with kinematic direction i ($i = 1, 2, \ldots, 6$) is defined as

$$m_{\alpha i}^{\text{eff}} = (\Gamma_{\alpha i})^2 m_\alpha \quad \text{(no sum on } \alpha).$$

If the effective masses of all modes are added in any global translational direction, the sum should give the total mass of the model (except for mass at kinematically restrained degrees of freedom). Thus, if the effective masses of the modes used in the analysis add up to a value that is significantly less than the model's total mass, this result suggests that modes that have significant participation in a certain excitation direction have not been extracted.

For coupled acoustic-structural eigenfrequency analysis, an additional acoustic effective mass is computed as outlined in *Coupled acoustic-structural medium analysis*.

Composite Modal Damping

Composite modal damping allows you to define a damping factor for each material or element in the model as a fraction of critical damping. These factors are then combined into a damping factor for each mode as weighted averages of the mass matrix associated with each material or element; when using the SIM architecture, you can

also include the weighted averages of the stiffness matrix. For more information, see *Defining Composite Modal Damping*.

Residual Modes

Several analysis types in Abaqus/Standard are based on the eigenmodes and eigenvalues of an undamped system. For example, in a mode-based steady-state dynamic analysis the mass, stiffness, and damping matrices and load vectors of the physical system are projected onto a set of eigenmodes. This projection results in a condensed (not necessarily diagonal) system of equations in terms of the modal amplitudes (or generalized degrees of freedom). The physical system response is obtained by scaling each eigenmode by its corresponding modal amplitude and superimposing the results (for more information, see *Linear dynamic analysis using modal superposition*).

Due to the computational cost, usually only a limited subset of the lower frequency eigenmodes of the system is extracted. Depending on the nature of the loading, the accuracy of the modal solution might suffer if too few higher frequency modes are used. To achieve sufficient accuracy using the available lower frequency eigenmodes, you can augment the modal subspace with additional modes known as *residual modes*. Residual modes help correct errors introduced by the modal basis truncation.

In Abaqus/Standard the residual mode vector is obtained from the linear static response of the finite element model to the nominal or scaled (for example, unit) load corresponding to the actual load applied in the mode-based analysis. The static solution is orthogonalized against the extracted eigenvectors and other residual mode vectors. The orthogonalization process automatically removes redundant linearly dependent vectors when constructing the basis of the modal subspace. As an outcome of the orthogonalization process, a pseudoeigenvalue corresponding to each residual mode is obtained as the Rayleigh ratio calculated for a residual mode shape vector.

Generally, the term eigenvalue is used in the Abaqus/Standard user assistance to refer to the actual eigenvalues and pseudoeigenvalues. All data (participation factors, etc.; see *Output*) associated with the modes (eigenmodes and residual modes) are ordered by increasing eigenvalue. Therefore, both eigenmodes and residual modes are assigned mode numbers. In the printed output in the data file, Abaqus/Standard clearly identifies which modes are eigenmodes and which modes are residual modes so that you can distinguish between them.

When generating dynamic substructures (see *Generating a Reduced Structural Damping Matrix for a Substructure*), residual modes usually provide the most benefit if the loading patterns defined in each of the load cases match the loading patterns defined under the corresponding substructure load cases in the substructure generation step.

By default, if you activate residual modes, all the calculated eigenmodes and residual modes are used in subsequent mode-based procedures, unless

- You choose to obtain a new set of eigenmodes and residual modes in a new frequency extraction step.
- You choose to select a subset of the available eigenmodes and residual modes in the mode-based procedure (selection of modes is described in each of the mode-based analysis type sections).

Residual modes cannot be calculated if the cyclic symmetric modeling capability is used.

Requesting Residual Modes Computation in an Eigenvalue Extraction Analysis

You can activate residual modes computation in an eigenfrequency extraction procedure. Abaqus/Standard supports residual mode extraction for all three eigensolvers, and several methods are available to specify the data (loads and their locations) defining the residual modes. *Table 2* indicates the eigensolver support for each of the methods.

Table 2: Eigensolver support for methods to request residual modes.

Method	Residual Mode Definition	Lanczos	AMS	Subspace
1	Load cases in eigenfrequency extraction procedure	Yes	Yes	Yes
2	Specifying degrees of freedom in eigenfrequency extraction procedure	Yes	Yes	Yes
3	Loads specified in mode-based procedures following eigenfrequency extraction	Yes	Yes	Yes
4	Static perturbation step preceding eigenfrequency extraction	Yes	No	Yes

Input File Usage: **FREQUENCY*, RESIDUAL MODES

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Basic**: **Include residual modes**

Requesting Residual Modes Computation Using the Lanczos and Subspace Iteration Eigensolvers

Both the Lanczos and subspace iteration eigensolvers offer several methods (described below) for defining residual modes. Methods 1 and 2 take precedence over Method 4, which takes precedence over Method 3. Therefore, if the frequency extraction step contains load cases that define residual modes or specified degrees of freedoms for residual vectors, the preceding static perturbation step (as well as subsequent linear dynamics procedures) has no effect on the computed residual modes. Similarly, if a static perturbation step is defined prior to the frequency extraction procedure and the frequency extraction step contains no load cases or specified degrees of freedom for residual vectors, Abaqus/Standard defines the residual modes by the loads only from the static perturbation step. The subspace iteration eigensolver can compute residual modes as the Lanczos eigensolver does. However, both solvers do not support residual modes computation for fully coupled structural-acoustic analysis.

Method 1 (Lanczos and Subspace Iteration): Load Cases in Eigenfrequency Extraction

You can specify residual modes using the loads defined in the load cases in the eigenfrequency extraction step. For each load case, Abaqus/Standard calculates two residual modes: one to include the effects of all in-phase (real) loads and another to include the effects of all out-of-phase (imaginary) loads. If you specify only real or only imaginary loads in the load case, Abaqus/Standard calculates a single residual mode. There are no fundamental restrictions for the loads used in the load cases. Abaqus/Standard supports all loads allowed in the modal analysis steps (the steady-state dynamic and transient modal dynamic analysis procedures). Using this method, you cannot specify loads outside the load cases and boundary conditions within the load cases.

An example for Method 1 (Lanczos and subspace iteration) is shown below.

```
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS|SUBSPACE, RESIDUAL MODES
…
*LOAD CASE
*CLOAD
*DLOAD
*END LOAD CASE
```

```
*LOAD CASE
*CLOAD
*DLOAD
*DSLOAD
*END LOAD CASE
…
*END STEP
```

Method 2 (Lanczos and Subspace Iteration): Specifying Degrees of Freedom in Eigenfrequency Extraction

You can request additional residual modes by specifying degrees of freedom. For both the Lanczos and subspace iteration eigensolvers, you can combine Method 2 with Method 1 or Method 3.

An example for Method 2 (Lanczos and subspace iteration) is shown below.

```
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS|SUBSPACE, RESIDUAL MODES
first data line
(additional data lines specifying degrees of freedom for residual modes)
first node label, first DOF label
second node label, second DOF label
… (no load cases defined)
*END STEP
```

Method 3 (Lanczos and Subspace Iteration): Loads Specified in Mode-Based Procedures Following Eigenfrequency Extraction

If load cases are not specified in the eigenfrequency extraction step and there is no static perturbation step immediately preceding the eigenfrequency extraction step, Abaqus/Standard computes residual modes for loads applied in the subsequent mode-based procedure specified in the same analysis job (not in the restart analysis). Abaqus/Standard calculates one residual mode for all real loads and another for all imaginary loads (if they are allowed) for each load case. Otherwise, Abaqus/Standard assumes that all loads are part of a single load case and calculates only one real-imaginary residual mode pair for that particular step. Abaqus/Standard takes into account all loads specified in the subsequent mode-based procedure. However, it ignores amplitude curve definitions. Therefore, if you associate different loads within the same load case with different amplitude curves, Abaqus/Standard does not automatically create a residual mode per a group of loads sharing the same amplitude.

An example for Method 3 (Lanczos and subspace iteration) is shown below.

```
(no preceding static perturbation step)
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS|SUBSPACE, RESIDUAL MODES
… (no load cases defined)
*END STEP
*STEP
*STEADY STATE DYNAMICS
…
*LOAD CASE
*CLOAD
*DLOAD
*END LOAD CASE
*LOAD CASE
*CLOAD
*DLOAD
*DSLOAD
*SLOAD
*END LOAD CASE
*END STEP
```

Method 4 (Lanczos and Subspace Iteration): Static Perturbation Step Preceding Eigenfrequency Extraction

If you do not specify load cases in the eigenfrequency extraction step, you can define residual modes by specifying loads in a static perturbation step immediately preceding the eigenfrequency extraction step. If you specify multiple load cases in this static linear perturbation analysis, Abaqus/Standard calculates one residual mode for each load case; otherwise, it assumes that all loads belong to a single load case and calculates only one residual mode. When you request residual modes, the boundary conditions applied in the frequency extraction step must match those applied in the preceding static perturbation step. In addition, in the immediately preceding static perturbation step, Abaqus/Standard requires the following conditions:

- If multiple load cases are used, the boundary conditions applied in each load case must be identical.
- The boundary condition magnitudes are zero.

An example for Method 4 (Lanczos and subspace iteration) is shown below.

```
*STEP, PERTURBATION
*STATIC
*LOAD CASE
*CLOAD
*DLOAD
*END LOAD CASE
*LOAD CASE
*CLOAD
*DLOAD
*DSLOAD
*SLOAD
*END LOAD CASE
*END STEP
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS|SUBSPACE, RESIDUAL MODES
... (no load cases defined)
*END STEP
```

Requesting Residual Modes Computation Using the AMS Eigensolver

The AMS eigensolver offers several methods for defining residual modes.

Method 1 (AMS): Load Cases in Eigenfrequency Extraction

Abaqus/Standard computes residual modes for the loads specified in the load cases in the eigenfrequency extraction step. It calculates one residual mode for each load case and for each loading type (real/imaginary). There are no fundamental restrictions for the loads used in the load cases. Abaqus/Standard supports all loads allowed in the modal analysis steps (the steady-state dynamic and transient modal dynamic analysis procedures). It takes the concentrated load magnitudes into account; therefore, each load case can specify loads combined from several differently scaled concentrated loads.

An example for Method 1 (AMS) is shown below.

```
*STEP
*FREQUENCY, EIGENSOLVER=AMS, RESIDUAL MODES
...
*LOAD CASE
*CLOAD
*CLOAD
*DLOAD
*END LOAD CASE
*LOAD CASE
*CLOAD
```

```
*CLOAD
*CLOAD
*DSLOAD
*END LOAD CASE
…
*END STEP
```

Method 2 (AMS): Specifying Degrees of Freedom in Eigenfrequency Extraction

You can request additional residual modes by specifying degrees of freedom. For the AMS eigensolver, you can combine Method 2 with Method 1 or Method 3.

An example for Method 2 (AMS) is shown below.

```
*STEP
*FREQUENCY, EIGENSOLVER=AMS, RESIDUAL MODES
first data line
(additional data lines specifying degrees of freedom for residual modes)
first node label, first DOF label
second node label, second DOF label
… (no load cases defined)
*END STEP
```

Method 3 (AMS): Loads Specified in Mode-Based Procedures Following Eigenfrequency Extraction

If you do not specify load cases in the frequency extraction step, Abaqus/Standard computes residual modes for all degrees of freedom at which concentrated loads are applied in the subsequent mode-based procedures specified in the same analysis job (not in the restart analysis). It computes a residual mode for every degree of freedom loaded with the concentrated load.

An example for Method 3 (AMS) is shown below.

```
*STEP
*FREQUENCY, EIGENSOLVER=AMS, RESIDUAL MODES
… (no load cases defined)
*END STEP
*STEP
*STEADY STATE DYNAMICS
…
*LOAD CASE
*CLOAD
*DLOAD
*END LOAD CASE
*LOAD CASE
*CLOAD
*DLOAD
*DSLOAD
*END LOAD CASE
*END STEP
*STEP
*MODAL DYNAMIC
*CLOAD
*DLOAD
*END STEP
```

Evaluating Frequency-Dependent Material Properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in the frequency extraction procedure. This evaluation is required because the stiffness cannot be modified during the eigenvalue extraction procedure. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness associated with frequency-dependent springs and dashpots at zero frequency and does not consider the stiffness contributions from frequency domain viscoelasticity. If you do specify a frequency, only the real part of the stiffness contributions from frequency domain viscoelasticity is considered.

Evaluating the properties at a specified frequency is particularly useful in analyses in which the eigenfrequency extraction step is followed by a subspace projection steady-state dynamic step (see *Subspace-Based Steady-State Dynamic Analysis*). In these analyses the eigenmodes extracted in the frequency extraction step are used as global basis functions to compute the steady-state dynamic response of a system subjected to harmonic excitation at a number of output frequencies. The accuracy of the results in the subspace projection steady-state dynamic step is improved if you choose to evaluate the material properties at a frequency in the vicinity of the center of the range spanned by the frequencies specified for the steady-state dynamic step.

Input File Usage: **FREQUENCY*, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: **Step->Create**: **Frequency**: **Other**: **Evaluate dependent properties at frequency**

Generating a Flexible Body

Abaqus/Standard can generate a flexible body from a natural frequency extraction for use in the Simpack flexible body dynamics solver. Only the `.sim` file results format is supported. Simpack converts all of the data and units into a Flexible Body Interface (FBI) file.

A flexible body in Simpack can be kinematically connected to other bodies of the multibody system by joints or connections that constrain the flexible body in certain degrees of freedom. You must use a boundary condition in the natural frequency extraction to constrain those degrees of freedom.

You can match the range of extracted frequencies to the use case in the flexible body dynamics simulation.

Input File Usage: **FLEXIBLE BODY*, TYPE=SIMPACK

Abaqus/CAE Usage: Flexible body generation is not supported in Abaqus/CAE.

Initial Conditions

If the frequency extraction procedure is the first step in an analysis, the initial conditions form the *base state* for the procedure (except for initial stresses, which cannot be included in the frequency extraction if it is the first step). Otherwise, the base state is the current state of the model at the end of the last general analysis step (*General and Perturbation Procedures*). Initial stress stiffness effects (specified either through defining initial stresses or through loading in a general analysis step) are included in the eigenvalue extraction only if geometric nonlinearity is considered in a general analysis procedure before the frequency extraction procedure.

If initial stresses must be included in the frequency extraction and there is not a general nonlinear step before the frequency extraction step, a “dummy” static step—which includes geometric nonlinearity and which maintains the initial stresses with appropriate boundary conditions and loads—must be included before the frequency extraction step.

Initial Conditions describes all the available initial conditions.

Boundary Conditions

Nonzero magnitudes of boundary conditions in a frequency extraction step are ignored; the degrees of freedom specified are fixed (*Boundary Conditions*).

Boundary conditions defined in a frequency extraction step are not used in subsequent general analysis steps (unless they are respecified).

In a frequency extraction step involving piezoelectric elements, the electric potential degree of freedom must be constrained at least at one node to remove numerical singularities arising from the dielectric part of the element operator.

Defining Primary and Secondary Bases for Modal Superposition Procedures

If displacements or rotations are to be prescribed in subsequent dynamic modal superposition procedures, boundary conditions must be applied in the frequency extraction step; these degrees of freedom are grouped into "bases." The bases are then used for prescribing motion in the modal superposition procedure—see *Transient Modal Dynamic Analysis*.

Boundary conditions defined in the frequency extraction step supersede boundary conditions defined in previous steps. Hence, degrees of freedom that were fixed before the frequency extraction step are associated with a specific base if they are redefined with reference to such a base in the frequency extraction step.

The Primary Base

By default, all degrees of freedom listed for a boundary condition are assigned to an unnamed "primary" base. If the same motion is prescribed at all fixed points, the boundary condition is defined only once; and all prescribed degrees of freedom belong to the primary base.

Unless removed in the frequency extraction step, boundary conditions from the last general analysis step become fixed boundary conditions for the frequency step and belong to the primary base.

If all rigid body motions are not suppressed by the boundary conditions that make up the primary base, you must apply a suitable frequency shift to avoid numerical problems.

Input File Usage: **BOUNDARY*

The **BOUNDARY* option without the BASE NAME parameter can appear only once in a frequency extraction step.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**

Secondary Bases

If the modal superposition procedure has more than one independent base motion, the driven nodes must be grouped together into "secondary" bases in addition to the primary base. The secondary bases must be named. (See *Base motions in modal-based procedures*.) Secondary bases are used only in modal dynamic and steady-state dynamic (not direct) procedures.

The degrees of freedom associated with secondary bases are not suppressed; instead, a "big" mass is added to each of them. To provide six digits of numerical accuracy, Abaqus/Standard sets each "big" mass equal to 10^6 times the total mass of the structure and each "big" rotary inertia equal to 10^6 times the total moment of inertia of the structure. Hence, an artificial low frequency mode is introduced for every degree of freedom in a secondary base. To keep the requested range of frequencies unchanged, Abaqus/Standard automatically increases the

number of eigenvalues extracted. Consequently, the cost of the eigenvalue extraction step increases as more degrees of freedom are included in the secondary bases. To reduce the analysis cost, keep the number of degrees of freedom associated with secondary bases to a minimum. This can sometimes be done by reducing several secondary bases that all have the same prescribed motion to a single node by using BEAM type MPCs (*General Multi-Point Constraints*).

For the Lanczos and subspace iteration methods a negative shift must be used with either the rigid body modes or secondary bases.

The "big" masses are not included in the model statistics, and the total mass of the structure and the printed messages about masses and inertia for the entire model are not affected. However, the presence of the masses is noticeable in the output tables printed for the eigenvalue extraction step, as well as in the information for the generalized masses and effective masses. See *Double cantilever subjected to multiple base motions* for an example of the use of the base motion feature.

More than one secondary base can be defined by repeating the boundary condition definition and assigning different base names.

Input File Usage: **BOUNDARY*, BASE NAME=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *frequency_step*; **Category: Mechanical**; **Types for Selected Step: Secondary base**; **Constrained degrees-of-freedom**: **Region**: *select region*, **U1**, **U2**, **U3**, **UR1**, **UR2**, and/or **UR3**

Controlling Damping for Low Frequency Modes

You can control damping of the low frequency modes (rigid body modes and low frequency modes with secondary base motions) in transient modal analyses. For more information, see *Controlling Damping*.

Loads

Applied loads (*About Loads*) are ignored during a frequency extraction analysis. If loads were applied in a previous general analysis step and geometric nonlinearity was considered for that prior step, the load stiffness determined at the end of the previous general analysis step is included in the eigenvalue extraction (*General and Perturbation Procedures*).

Predefined Fields

Predefined fields cannot be prescribed during natural frequency extraction.

Material Options

The density of the material must be defined (*Density*). The following material properties are not active during a frequency extraction: plasticity and other inelastic effects, rate-dependent material properties, thermal properties, mass diffusion properties, electrical properties (although piezoelectric materials are active), and pore fluid flow properties—see *General and Perturbation Procedures*.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement or acoustic elements in Abaqus/Standard (including those with temperature, pressure, or electrical degrees of freedom) can be used in a frequency extraction procedure.

Output

The eigenvalues (EIGVAL), eigenfrequencies in cycles/time (EIGFREQ), generalized masses (GM), composite modal damping factors (CD), participation factors for displacement degrees of freedom 1–6 (PF1–PF6) and acoustic pressure (PF7), and modal effective masses for displacement degrees of freedom 1–6 (EM1–EM6) and acoustic pressure (EM7) are written automatically to the output database as history data. Output variables such as stress, strain, and displacement (which represent mode shapes) are also available for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values.

The eigenvalues and corresponding frequencies (in both radians/time and cycles/time) are also listed automatically in the printed output file, along with the generalized masses, composite modal damping factors, participation factors, and modal effective masses.

The AMS eigensolver does not compute composite modal damping factors or modal effective masses. In addition, you cannot request output to the results (`.fil`) file.

You can restrict output to the results, data, and output database files by selecting the modes for which output is desired (see *Output to the Data and Results Files* and *Output to the Output Database*).

Input File Usage: Use one of the following options:

**EL FILE*, MODE, LAST MODE
**EL PRINT*, MODE, LAST MODE
**OUTPUT*, MODE LIST

Abaqus/CAE Usage: Step module: **Output->Field Output Requests->Create**: **Frequency**: **Specify modes**

Energy Output

Several types of energy output variables are supported in eigenvalue extraction procedures. They include the elastic strain energy density (SENER), whole element variables (ELKE and ELSE), whole element energy density variables (EKEDEN and ESEDEN), and whole and partial model variables (ALLKE and ALLSE). The values of all of these energy variables, except SENER, are normalized. Normalization is performed for each eigenmode separately, such that the kinetic and strain energies add up to one for the whole model. Computation of the energy variables is implemented for all but residual and singular acoustic modes.

The energy variables that can be written to the output database are defined in *Total Energy Output Quantities*.

Input File Usage: **ENERGY OUTPUT*
list of output variables

Abaqus/CAE Usage: Step module: history output request editor: **Select from list below**

Eigenvalue Extraction Results Postprocessing

You can use a restart analysis to extract the nodal and element variables from the natural frequency extraction data (see *Recovering Additional Results Output from Restart Data in Abaqus/Standard*). This capability enables Abaqus/Standard to extract the eigenmodes from the analysis database and calculate the values of the output variables without recomputing the eigenmodes and eigenfrequencies. Using it can be especially beneficial for large models when you need additional output from the natural frequency extraction step. However, you cannot use this capability to output energy and energy-related variables.

Input File Usage: **POST OUTPUT*, STEP=*frequency_extraction_step_number*

Abaqus/CAE Usage: Recovering additional results from an eigenvalue extraction procedure is not supported in Abaqus/CAE.

Input File Template

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
**
*STEP (,NLGEOM)
If NLGEOM is used, initial stress and preload stiffness effects
are included in the frequency extraction step
*STATIC
…
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*BOUNDARY
Data lines to specify zero-valued or nonzero boundary conditions
*END STEP
**
*STEP, PERTURBATION
*STATIC
…
*LOAD CASE, NAME=load case name
Keywords and data lines to define loading  for this load case
*END LOAD CASE
…
*END STEP
**
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS, RESIDUAL MODES
Data line to control eigenvalue extraction
*BOUNDARY

*BOUNDARY, BASE NAME=name
Data lines to assign degrees of freedom to a secondary base
*END STEP
```

Complex Eigenvalue Extraction

Products: Abaqus/Standard Abaqus/CAE

Overview

The complex eigenvalue extraction procedure:

- performs eigenvalue extraction to calculate the complex eigenvalues and the corresponding complex mode shapes of a system;
- is a linear perturbation procedure;
- requires that an eigenfrequency extraction procedure (*Natural Frequency Extraction*) be performed prior to the complex eigenvalue extraction;
- can use the high-performance SIM software architecture (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*);
- will include initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the base state step definition (*General and Perturbation Procedures*);
- can include friction, damping, and unsymmetric load stiffness contributions;
- can include unsymmetric damping and stiffness contributions in acoustic finite elements due to underlying mean flow (*Coupled Acoustic-Structural Analysis*); and
- cannot be used in a model defined as a cyclic symmetric structure (*Analysis of Models that Exhibit Cyclic Symmetry*).

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- **COMPLEX FREQUENCY*
- *Configuring a complex frequency procedure*
- *Output to the Data and Results Files*
- *Output to the Output Database*

Complex Eigenvalue Extraction

The complex eigenvalue extraction procedure uses a projection method to extract the complex eigenvalues of the current system. The eigenvalue problem of the finite element model is formulated in the following manner:

$$\left(\mu^2 M^{MN} + \mu C^{MN} + K^{MN}\right) \phi_R^N = 0,$$

where

M^{MN}

is the mass matrix (which is symmetric and, in general, is semi-positive definite);

C^{MN}

is the damping matrix;

K^{MN}

is the stiffness matrix (which can include initial stress stiffness and friction effects and, therefore, in general is unsymmetric);

μ

is the complex eigenvalue;

ϕ_R^N

is the right complex eigenvector;

ϕ_L^N

is the left complex eigenvector and is defined as follows:

$$\left(\phi_L^N\right)^* \left(\mu^2 M^{MN} + \mu C^{MN} + K^{MN}\right) = 0;$$

where $\left(\phi_L^N\right)^*$ is a transpose conjugate left eigenvector

M and N

are degrees of freedom.

The complex eigenvalue extraction procedure in Abaqus/Standard uses a subspace projection method; thus, the eigenmodes of the undamped system with the symmetrized stiffness matrix must be extracted using the eigenfrequency extraction procedure prior to the complex eigenvalue extraction step. By default, the entire subspace is used as the base vector; this subspace can be reduced as described below. Abaqus/Standard always computes all the complex eigenmodes available in the projection subspace (taking into account any user-specified modifications to the subspace). The user-specified number of requested eigenmodes and frequency range for the complex eigenvalue extraction procedure do not influence the number of computed complex eigenmodes. It determines only the number of reported modes, which cannot be higher than the dimension of the projected subspace. To modify the number of computed eigenmodes, reduce the projection subspace as described below or change the number of eigenmodes extracted in the prior natural frequency extraction step accordingly. If you do not specify the number of requested complex modes or the frequency range, all the computed modes will be reported.

To take into account the unsymmetric effects, the unsymmetric matrix solution and storage scheme is used automatically for a complex eigenvalue extraction step. The unsymmetric effects will be disregarded if you specify that the symmetric solution and storage scheme should be used (see *Defining an Analysis*).

Input File Usage: **COMPLEX FREQUENCY*
number of complex eigenmodes, frequency_min, frequency_max

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Number of eigenvalues requested: All** or **Value**, **Minimum frequency of interest (cycles/time):** *value*, **Maximum frequency of interest (cycles/time):** *value*

Shift Point

You can specify a shift point, S, in cycles per time, for the complex eigenvalue extraction procedure ($S \geq 0$).

Abaqus/Standard reports the complex eigenmodes, μ_n, in order of increasing $|Im(\mu_n) - S|$ so that the modes with the imaginary part closest to a given shift point are reported first. This feature is useful when a particular frequency range is of concern. The default is no shift.

Input File Usage: **COMPLEX FREQUENCY*
, , , S

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Frequency shift (cycles/time):** S

Normalization

For complex eigenvalue extraction analysis both displacement and modal complex eigenvector normalization are available. Displacement normalization is the default in SIM-based analysis. Modal normalization is the only option available if the SIM-based architecture is not used.

If displacement normalization is selected, the complex eigenvectors are normalized so that the largest value in each vector is unity and the imaginary part is zero. If modal normalization is selected, only the complex eigenvectors of the projected system (GU) are normalized using the displacement method and no normalization of the complex eigenvectors in the finite element subspace is performed. For large eigenproblems the displacement normalization can become time consuming; therefore, modal normalization is recommended.

Input File Usage: Use the following option to select displacement normalization (available only if the SIM-based architecture is used):

**COMPLEX FREQUENCY*, NORMALIZATION=DISPLACEMENT

Use the following option to select modal normalization (the only option if the SIM-based architecture is not used):

**COMPLEX FREQUENCY*, NORMALIZATION=MODAL

Abaqus/CAE Usage: You cannot select the normalization method of the complex eigenvectors in Abaqus/CAE; the default method is used.

Selecting the Eigenmodes on Which to Project

You can select eigenmodes of the undamped system with the symmetrized stiffness matrix on which the subspace projection will be performed. You can select them by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the eigenmodes that belong to specified frequency ranges. If you do not select the eigenmodes, all modes extracted in the prior eigenfrequency extraction step are used in the modal superposition.

Input File Usage: Use one of the following options to select the eigenmodes by specifying mode numbers:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS
**SELECT EIGENMODES*, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to define the eigenmodes by specifying a frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the eigenmodes in Abaqus/CAE; all modes extracted are used in the subspace projection.

Evaluating Frequency-Dependent Material Properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in the complex eigenvalue extraction procedure. This evaluation is necessary because the operators cannot be modified during the eigenvalue extraction procedure. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness and damping associated with frequency-dependent springs and dashpots at zero frequency and does not consider the stiffness and damping contributions from frequency-domain viscoelasticity. If you do specify a frequency, the stiffness and damping contributions from frequency-domain viscoelasticity are considered.

Input File Usage: **COMPLEX FREQUENCY*, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: **Create Step**: **Complex Frequency**: **Other**: **Evaluate dependent properties at frequency:** *value*

Right and Left Complex Eigenvectors

For complex eigenvalue extraction analysis right or left complex eigenvectors can be requested. By default, the right eigenvectors are extracted. The left eigenvectors are available only in analyses that are based on the SIM architecture. You can extract both right and left complex eigenvectors in the same analysis, but they must be requested in separate steps. You should select modal normalization of the complex eigenvectors if you want to extract both the right and left eigenvectors.

Input File Usage: Use the following option to extract the right complex eigenvectors:

**COMPLEX FREQUENCY*, RIGHT EIGENVECTORS (default, the only option if the SIM architecture is not used)

Use the following option to extract the left complex eigenvectors:

**COMPLEX FREQUENCY*, LEFT EIGENVECTORS (only if the SIM architecture is used)

Abaqus/CAE Usage: Only the right complex eigenvectors are extracted in Abaqus/CAE.

Contact Conditions with Sliding Friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom will not be constrained and the effect of friction will result in an unsymmetric contribution to the stiffness matrix. At other nodes in contact the tangential degrees of freedom will be constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis.

The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see *Coulomb friction*. The complex eigensolver allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: **COMPLEX FREQUENCY*, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Include friction-induced damping effects**

Damping

In complex eigenvalue extraction analysis damping can be defined by dashpots (see *Dashpots*), by "Rayleigh" damping associated with materials and elements (see *Material Damping*), and by quiet boundaries on infinite elements or acoustic elements. In addition, as described in *Contact Conditions with Sliding Friction* above, friction-induced damping can be included.

Structural damping, damping contributions from frequency-domain viscoelasticity, and all types of modal damping (except composite modal damping) are supported in complex eigenvalue extraction using the high-performance SIM architecture.

Prescribing Motion, Transport Velocity, and Acoustic Flow Velocity

Motion, transport velocity, and acoustic flow velocity affect complex frequency analyses. Motion and transport velocity must be specified in a preceding steady-state transport general step, and their effects are included in the complex frequency step. The acoustic flow velocity has no effect in steady-state transport steps, and acoustic flow velocities specified in a steady-state transport step are not propagated to perturbation steps. The acoustic flow velocity must be specified in each linear perturbation step where it is desired.

Stiffness Projection Performance

Two methods are available to reduce the computational time in the stiffness projection phase of the analysis, but they should be used with caution. By default, the complex eigensolver projects the stiffness operator on the modal subspace. This is necessary when advanced mechanical effects, such as friction-induced effects and frequency-dependent properties, must be considered. However, for large modal subspaces, this operation can be expensive.

Skipping Projection of the Stiffness Operator

If advanced mechanical effects can be ignored, you can instruct the complex eigensolver to skip projection of the stiffness operator on the modal subspace. In this case the modal stiffness operator is defined by the diagonal matrix of the real eigenvalues corresponding to the modal subspace basis vectors.

Input File Usage: **COMPLEX FREQUENCY*, STIFFNESS PROJECTION=NO

Abaqus/CAE Usage: Skipping projection of the stiffness operator is not supported in Abaqus/CAE.

Specifying That Advanced Mechanical Properties Are Localized in the Contact Area Only

You can specify that advanced mechanical properties (for example, friction) are localized in the contact area only and the rest of the model has the same properties as in the preceding real eigenfrequency extraction analysis. In this case the modal stiffness operator is combined from the diagonal matrix of the real eigenvalues corresponding

to the modal subspace basis vectors and the projection of the additional contact stiffness terms on the modal subspace.

Input File Usage: **COMPLEX FREQUENCY*, STIFFNESS PROJECTION=CONTACT ONLY

Abaqus/CAE Usage: Specifying that advanced mechanical properties are localized in the contact area only is not supported in Abaqus/CAE.

Initial Conditions

Initial conditions cannot be specified for complex eigenvalue extraction.

Boundary Conditions

Boundary conditions cannot be defined during complex eigenvalue extraction. The boundary conditions will be the same as in the prior natural frequency extraction analysis.

Loads

Applied loads (*About Loads*) are ignored during a complex eigenvalue extraction. If loads were applied in a previous general analysis step in which nonlinear geometric effects were included, the load stiffness determined at the end of the previous general analysis step is included in the complex eigenvalue extraction (see *General and Perturbation Procedures*).

Coriolis distributed loading adds an unsymmetric contribution to the damping operator, which is currently accounted for only in solid and truss elements.

Predefined Fields

Predefined fields cannot be prescribed during complex eigenvalue extraction.

Material Options

The density of the material must be defined (see *Density*). The following material properties are not active during complex eigenvalue extraction:

- plasticity and other inelastic effects;
- rate-dependent material properties, excluding friction, which can be rate dependent if the velocity differential on the contact interface exists;
- thermal properties;
- mass diffusion properties;
- electrical properties (although piezoelectric materials are active); and
- pore fluid flow properties.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in complex eigenvalue extraction.

Output

The real (EIGREAL) and imaginary (EIGIMAG) parts of the eigenvalues, (α and ω); frequencies in cycles/time (EIGFREQ); and effective damping ratios (DAMPRATIO = $-2 * \alpha / |\omega|$) are written automatically to the data (`.dat`) file and to the output database (`.odb`) file as history data. In addition, you can request that the generalized displacements (GU), which are the modes of the projected system, be written to the output database file (see *Output to the Output Database*). Output variables such as stress, strain, and displacement (which represent mode shapes) are also available for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values.

The only energy density available in eigenvalue extraction procedures is the elastic strain energy density, SENER. All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

You can restrict output to the data file and output database file by selecting the modes for which output is desired (see *Output to the Data and Results Files* or *Output to the Output Database*). Output to the results (`.fil`) file is not available for the complex eigenvalue extraction procedure.

Setting the Cutoff Value for Complex Eigenmodes

You can also set the cutoff value for complex eigenmodes, so only complex modes with the real part of the eigenvalue higher than the cutoff value are written to the output database file. The default cutoff value is 0.0. If the cutoff value is not set, all complex modes are output.

Input File Usage: Use one of the following options to select complex eigenmodes for output:

**COMPLEX FREQUENCY*, UNSTABLE MODES ONLY
**COMPLEX FREQUENCY*, UNSTABLE MODES ONLY=*value*

The SIM Architecture

The complex eigenvalue extraction analysis can be performed using the SIM architecture. The advantages of performing the complex eigenvalue extraction procedure using the SIM architecture are as follows:

- structural damping, including damping defined with viscoelastic material, is taken into account;
- modal damping can be specified;
- matrices representing the stiffness, mass, and damping can be defined (both symmetric and unsymmetric matrices are supported); and
- the AMS eigensolver can be used to generate the projection subspace for the complex eigenvalue extraction.

When the AMS eigensolver is used for computing the projection subspace, you should increase the accuracy of the AMS eigensolution by increasing the values of the AMS parameters and by increasing the highest frequency of interest.

Input File Template

```
*HEADING
…
*SURFACE INTERACTION
*FRICTION
Specify zero friction coefficient
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
```

```
Data lines to specify initial conditions
**
*STEP (,NLGEOM)
If NLGEOM is used, initial stress and preload stiffness effects
will be included in the eigenvalue extraction steps
*STATIC
…
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*BOUNDARY
Data lines to specify zero-valued or nonzero boundary conditions
*END STEP
**
*STEP(,NLGEOM)
*STATIC
Data line to define incrementation
*CHANGE FRICTION
*FRICTION
Data lines to redefine friction coefficient
*MOTION, ROTATION or TRANSLATION
Data lines to define the velocity differential
*END STEP
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*END STEP
**
*STEP
*COMPLEX FREQUENCY
Data line to control complex eigenvalue extraction
*SELECT EIGENMODES
Data lines to define applicable mode ranges
*END STEP
```

Transient Modal Dynamic Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A modal dynamic analysis:

- is a linear perturbation procedure;
- is used to analyze transient linear dynamic problems using modal superposition;
- can be performed only after a frequency extraction procedure since it bases the structure's response on the modes of the system; and
- can include modal damping, global damping, and damping effects from material or element damping.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Dynamic Analysis Procedures*
- **MODAL DYNAMIC*
- *Configuring a modal dynamics procedure*

Modal Dynamic Analysis

Transient modal dynamic analysis gives the response of the model as a function of time based on a given time-dependent loading. The structure's response is based on a subset of the modes of the system, which must first be extracted using an eigenfrequency extraction procedure (*Natural Frequency Extraction*). The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

The modal amplitudes are integrated through time, and the response is synthesized from these modal responses. For linear systems the modal dynamic procedure is much less expensive computationally than the direct integration of the entire system of equations performed in the dynamic procedure (*Implicit Dynamic Analysis Using Direct Integration*).

As long as the system is linear and is represented correctly by the modes being used (which are generally only a small subset of the total modes of the finite element model), the method is also very accurate because the integration operator used is exact whenever the forcing functions vary piecewise linearly with time. You should ensure that the forcing function definition and the choice of time increment are consistent for this purpose. For example, if the forcing is a seismic record in which acceleration values are given every millisecond and it is assumed that the acceleration varies linearly between these values, the time increment used in the modal dynamic procedure should be a millisecond.

The user-specified maximum number of increments is ignored in a modal dynamic step. The number of increments is based on both the time increment and the total time chosen for the step.

While the response in this procedure is for linear vibrations, the prior response can be nonlinear and stress stiffening (initial stress) effects will be included in the response if nonlinear geometric effects were included in

the step definition for the base state of the eigenfrequency extraction procedure, as explained in *Natural Frequency Extraction*.

Selecting the Modes and Specifying Damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the Modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS
**SELECT EIGENMODES*, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying Modal Damping

Damping is almost always specified for a mode-based procedure; see *Material Damping*. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

**MODAL DAMPING*, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

**MODAL DAMPING*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Damping**

Example of Specifying Damping

Figure 1 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
f1, d1
f2, d2
f2, d3
f3, d3
f4, d4
```

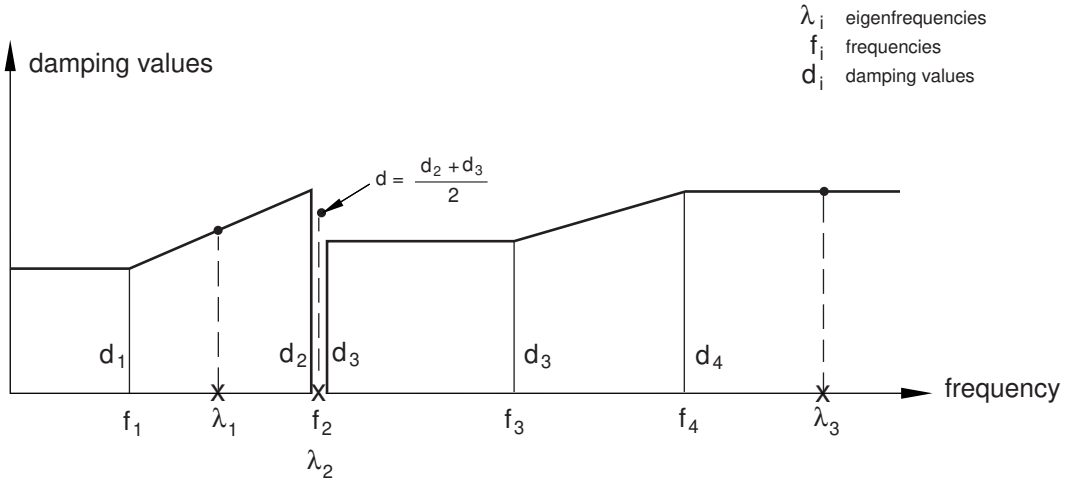


Figure 1: Damping coefficients specified by frequency range.

Rules for Selecting Modes and Specifying Damping Coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Specifying Global Damping

For convenience you can specify constant global damping factors for all selected eigenmodes for mass and stiffness proportional viscous factors, as well as stiffness proportional structural damping. Structural damping is a commonly used damping model that represents damping as complex stiffness. This representation causes no difficulty for frequency domain analysis such as steady-state dynamics for which the solution is already

complex. However, the solution must remain real-valued in the time domain. To allow users to apply their structural damping model in the time domain, a method has been developed to convert structural damping to an equivalent viscous damping. This technique was designed so that the viscous damping applied in the frequency domain is identical to the structural damping if the projected damping matrix is diagonal. For further details, see *Modal dynamic analysis*.

Input File Usage: **GLOBAL DAMPING*, ALPHA=*factor*, BETA=*factor*, STRUCTURAL=*factor*

Abaqus/CAE Usage: Defining damping by global factors is not supported in Abaqus/CAE.

Material Damping

Structural and viscous material damping (see *Material Damping*) is taken into account in a transient modal analysis. The projection of damping operators onto the mode shapes is performed only one time during the frequency extraction step. If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

Controlling Damping

You can deactivate the structural or viscous damping in a transient modal procedure if desired.

You can also control damping of the low frequency eigenmodes in transient modal analyses. This control is useful for free structures and models with secondary base motions, and it controls all sources of damping including the modal damping. To include low frequency eigenmodes, set the low frequency cutoff value to a small negative value. To exclude them, either provide a low frequency cutoff value or allow Abaqus to calculate it; it will be six orders of magnitude smaller than the eigenfrequency of the first deformable eigenmode.

Input File Usage: Use the following option to deactivate structural and viscous damping in a specific transient modal dynamic step:

**DAMPING CONTROLS*, STRUCTURAL=NONE, VISCOUS=NONE

Use the following option to control damping of the low frequency eigenmodes in a specific transient modal dynamic step:

**DAMPING CONTROLS*, LOW FREQUENCY CUTOFF

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Transient Modal Dynamic Solver

The basic concept of the modal superposition method is that the equations of motion of the finite element model are projected onto the modal subspace. Solving the projected modal equations of motion yields the modal solution: generalized displacements, velocities, and accelerations as functions of time. The response of the structure is expressed in terms of the mode shape vectors and the modal solution.

If the finite element model is not damped or if only modal and/or global damping are specified:

- the projected modal damping operator is diagonal, and
- the modal equations of motion are uncoupled.

Abaqus/Standard uses the diagonal transient modal dynamic solver in that case. The modal response is integrated through time: Abaqus/Standard calculates the exact solution of the uncoupled differential equations of motion

assuming that all the loads vary piecewise linearly within each time increment. This calculation is very fast and stable.

If damping is introduced at the finite element level, the modal damping operator might not be diagonal. For example, when damping is specified as the material property or when dashpots or connectors with damping are used in the model, the damping operator is nondiagonal. When the modal damping operator is nondiagonal, Abaqus/Standard calculates the modal solution using either the state space or the conventional solver.

Abaqus/Standard selects the transient modal dynamic solver automatically and uses the diagonal solver whenever possible. For analyses with nondiagonal modal damping, Abaqus/Standard uses the state space solver by default. You can overwrite this default by selecting the conventional solver. In general, this is not recommended, unless the number of modes is very high and the state space solver setup phase is too costly. In addition, using the conventional solver for models with high levels of damping is not recommended.

State Space Transient Modal Dynamic Solver

The state space transient modal dynamic solver calculates exact modal solution for systems with arbitrary damping for loads that vary piecewise linearly with time. This solver uses the complex eigenmodes of the unsymmetric modal state space operator to represent the real transient modal solution. Obtaining the solution includes two phases:

- solving the unsymmetric eigenvalue problem for the state space operator at the “setup” phase and
- calculating the transient solution at the “apply” phase.

The setup phase is the most computationally intensive part of the analysis. It can be costly for systems with many thousands of modes. The calculated solution is more accurate than that obtained using the conventional solver, and the algorithm is stable.

The state space solver supports structural-acoustic transient modal analysis using the modal subspace built from the coupled structural-acoustic eigenmodes or the uncoupled structural and acoustic eigenmodes.

Input File Usage: Use the following option to select the state space transient modal dynamic solver (default):

**MODAL DYNAMIC*, SOLVER=STATE SPACE

Abaqus/CAE Usage: The state space transient modal dynamic solver is the default solver in Abaqus/CAE.

Conventional Transient Modal Dynamic Solver

The conventional transient modal dynamic solver is based on the assumption that the nondiagonal entries of the modal damping operator vary linearly within each time increment together with the loads. The conventional solver calculates an approximate modal solution.

The solution accuracy depends on the time step size, and the time integration algorithm is not unconditionally stable. It can lose computational stability for models with high levels of damping. Accuracy and stability of the conventional solver are usually acceptable for common engineering simulations. However, obtaining accurate and stable solutions for specific cases might require using very small time increments.

The conventional solver does not support coupled structural-acoustic analysis.

Input File Usage: Use the following option to select the conventional transient modal dynamic solver:

**MODAL DYNAMIC*, SOLVER=CONVENTIONAL

Abaqus/CAE Usage: The conventional transient modal dynamic solver is not supported in Abaqus/CAE.

Initial Conditions

By default, the modal dynamic step will begin with zero initial displacements. If initial velocities have been defined (*Initial Conditions*), they will be used; otherwise, the initial velocities will be zero.

Alternatively, you can force the modal dynamic step to carry over the initial conditions from the immediately preceding step, which must be either another modal dynamic step or a static perturbation step:

- In most cases if the immediately preceding step is a modal dynamic step, both the displacements and velocities are carried over from the end of that step and used as initial conditions for the current step.
- If the immediately preceding step is a static perturbation step, the displacements are carried over from that step. If initial velocities have been defined (*Initial Conditions*), they will be used; otherwise, the initial velocities will be zero.

Input File Usage: Use the following option to begin the modal dynamic step with zero initial displacements:

**MODAL DYNAMIC*, CONTINUE=NO

Use the following option to force the modal dynamic step to carry over the initial conditions from the immediately preceding step:

**MODAL DYNAMIC*, CONTINUE=YES

Abaqus/CAE Usage: Use the following option to begin the modal dynamic step with zero initial displacements:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Basic**: **Zero initial conditions**

Use the following option to force the modal dynamic step to carry over the initial conditions from the immediately preceding step:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Basic**: **Use initial conditions**

Boundary Conditions

It is not possible to prescribe nonzero displacements and rotations (or acoustic pressure) directly as boundary conditions (*Boundary Conditions*) in mode-based dynamic response procedures. In these procedures the motion for nodes can be specified only as base motion, as described below. Nonzero displacement or acceleration history definitions given as boundary conditions are ignored in modal superposition procedures, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors.

Prescribed Motions in Modal Superposition Procedures

Boundary conditions must be applied during the eigenfrequency extraction step to the degrees of freedom that will be prescribed in the modal dynamic procedure. These degrees of freedom are grouped into one or more "bases" (see *Natural Frequency Extraction*). The unnamed base is called the "primary" base. Named "secondary" bases must be defined by specifying boundary conditions in the frequency extraction step. A different motion can be prescribed for each base.

The far-field nodes of infinite elements are always added to the primary base in modal procedures. The far-field nodes are added to the primary base even if they are not explicitly constrained via boundary conditions.

Specifying Primary or Secondary Base Motion

You specify primary base motion by defining a base motion without referring to a base. If the base motion is to be applied to a secondary base, it must refer to the name of the base defined in the eigenfrequency extraction step. You can prescribe secondary base motions in local or global (default) coordinate systems. If local coordinate systems are used, you must specify the directions of the secondary base motion in accordance with the applied nodal transformations.

For secondary base motion, you can also use degree of freedom 8 as the acoustic pressure degree of freedom. Acoustic pressure variation can be used in secondary bases only with big mass scaling. In such cases the reciprocal of the acoustic bulk modulus is used to scale the big mass value. To specify an acoustic pressure, use the displacement base motion; to specify the first and the second derivative of acoustic pressure, use the velocity and acceleration base motion, respectively.

Input File Usage: **BASE MOTION*, DOF=*n*, AMPLITUDE=*name*, BASE NAME=*secondary base*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; toggle on **Secondary base:** *boundary_condition_name*

Example

To illustrate the concept of primary and secondary bases, consider a single-bay frame with supports at nodes 1 and 4. If the input prior to the eigenfrequency extraction step includes the following boundary conditions:

- degrees of freedom 1 through 6 constrained at node 1
- degree of freedom 1 constrained at node 4
- degrees of freedom 3 through 6 constrained at node 4

and different base motions are assigned to degree of freedom 2 at nodes 1 and 4, the following step definitions could be used:

- an eigenfrequency extraction step that includes a boundary condition associated with `BASE2` constraining degree of freedom 2 at node 4; and
- a modal dynamic step that includes two base motion definitions: the primary base motion assigned to degree of freedom 2 that does not refer to a base and the secondary base motion assigned to degree of freedom 2 that refers to `BASE2`.

If boundary conditions were not given prior to the eigenfrequency extraction step, you would have to define them in the eigenfrequency extraction step. Again, the secondary base would be defined by a boundary condition with a base name.

Specifying the Degree of Freedom and the Time History of the Motion

You prescribe the displacements and rotations that are associated with boundary condition nodes during the modal dynamic response procedure. The base motions are fully defined by at most three global translations and three global rotations. Thus, at most one base motion can be defined for each translation and rotation component.

Primary base motions are always specified in global directions, regardless of the use of nodal transformations. You specify the degree of freedom in the global direction (1–6) for which the base motion is being defined. By

default, rotational primary base motion is defined with respect to the origin of the global coordinate system. You can specify separate centers of rotation for each rotational primary base motion.

You define secondary base motion by specifying named boundary conditions in the eigenvalue extraction analysis. You apply the base motion either in the global coordinate system or in a local coordinate system. To specify secondary base motion in a global direction at a node at which a coordinate transformation is applied, you must specify all three translational degrees of freedom (1–3) or all three rotational degrees of freedom (4–6) in the named boundary condition in the frequency extraction procedure if the base motion is translational or rotational, respectively.

The time history of a motion must be defined by an amplitude curve (*Amplitude Curves*).

Input File Usage: Use the following option to specify the degree of freedom and time history of the motion:

**BASE MOTION*, DOF=*n*, AMPLITUDE=*name*

Use the following option to apply the secondary base motion in the global coordinate system (default):

**BASE MOTION*, DOF=*n*, BASE NAME=*base name*, DIRECTION=GLOBAL

Use the following option to apply the secondary base motion in a local coordinate system:

**BASE MOTION*, DOF=*n*, BASE NAME=*base name*, DIRECTION=LOCAL

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom:** **U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Base motions are always applied in the global coordinate system in Abaqus/CAE.

Scaling the Amplitude of the Base Motion

The amplitude curve used to define the time history of the motion can be scaled. By default, the scaling factor is 1.0.

Input File Usage: **BASE MOTION*, DOF=*n*, AMPLITUDE=*name*, SCALE=*n*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom:** **U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*; **Amplitude scale factor:** *n*

Specifying the Type of Base Motion

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. Furthermore, if the displacement or velocity histories have nonzero initial values, Abaqus/Standard will make

corrections to the initial accelerations as described in *Modal dynamic analysis*. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

Input File Usage: Use one of the following options:

**BASE MOTION*, DOF=*n*, AMPLITUDE=*name*, TYPE=ACCELERATION
**BASE MOTION*, DOF=*n*, AMPLITUDE=*name*, TYPE=VELOCITY
**BASE MOTION*, DOF=*n*, AMPLITUDE=*name*, TYPE=DISPLACEMENT

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**

Calculating the Response of the Structure

The degrees of freedom associated with the primary base are set to zero in the eigenfrequency extraction step, and primary base motions are introduced by multiplying the base acceleration with the modal participation factors. Hence, Abaqus/Standard calculates the response of the structure with respect to the primary base. If the rotational degrees of freedom are references in the primary base motion definition, the rotation is defined, as default, about the origin of the coordinate system unless you provide the center of rotation.

To implement secondary base motion for mechanical degrees of freedom (1–6), Abaqus/Standard uses a penalty method called the "big mass" method. For the pressure degree of freedom, a similar technique is employed, but instead of using mass, the reciprocal of the acoustic bulk modulus is used. The degrees of freedom associated with the secondary bases are not set to zero in the eigenfrequency extraction step; instead, a “big” mass is added to each of them. Any degree of freedom in a secondary base that was constrained by a regular boundary condition in a previous general step will be released, and a big mass will be added to that degree of freedom. Secondary base motions are introduced by nodal forces, obtained by multiplying the base acceleration (or acoustic pressure) with the big mass (or reciprocal of the acoustic bulk modulus). Although the secondary base motions are defined in absolute terms, the response calculated at the secondary bases is relative to the motion of the primary base for the translational degrees of freedom. The rotational secondary bases are defined about the nodes included in the node sets specified in the base name definition. Therefore, you cannot change the center of rotation for secondary bases.

For a more detailed description of the base motion procedure, see *Base motions in modal-based procedures*.

Loads

The following loads can be prescribed in modal dynamic analysis, as described in *Concentrated Loads*:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Predefined Fields

Predefined temperature fields are not allowed in transient modal dynamic analysis. Other predefined fields are ignored.

Material Options

The density of the material must be defined (*Density*). The following material properties are not active during a modal dynamic analysis: plasticity and other inelastic effects, rate-dependent material properties, thermal

properties, mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties. See *General and Perturbation Procedures*.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature and pressure degrees of freedom) can be used in a modal dynamic analysis.

Output

All the output variables in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*. The values of nodal solution variables U, V, and A in modal dynamics in the time domain are relative to the motion of the primary base. Hence, the sum of the relative motion and the base motion of the primary base yields the total motion; this total motion is available by requesting output variables TU, TV, and TA. In the absence of primary base motions, the relative and total motions are identical.

The following modal variables can be output to the data or results files (see *Output to the Data and Results Files*):

GU	Generalized displacements for all modes.
GV	Generalized velocities for all modes.
GA	Generalized accelerations for all modes.
SNE	Elastic strain energy for the entire model per each mode.
KE	Kinetic energy for the entire model per each mode.
T	External work for the entire model per each mode.
BM	Base motion.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in modal dynamic analysis. However, whole model variables such as ALLIE (total strain energy) are available for mode-based procedures as output to the data or results files (see *Output to the Data and Results Files*).

The computational expense of a modal dynamic analysis can be decreased significantly by reducing the amount of output requested.

Input File Template

```
*HEADING
...
*AMPLITUDE, NAME=amplitude
Data lines to define amplitude variations
**
*STEP
*FREQUENCY
Data line to specify the number of modes to be extracted
*BOUNDARY
```

```
Data lines to assign degrees of freedom to the primary base
*BOUNDARY, BASE NAME=base
Data lines to assign degrees of freedom to a secondary base
*END STEP
**
*STEP
*MODAL DYNAMIC
Data line to control time incrementation
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data line to define modal damping
*BASE MOTION, DOF=dof, AMPLITUDE=amplitude
*BASE MOTION, DOF=dof, AMPLITUDE=amplitude, BASE NAME=base
*END STEP
```

Mode-Based Steady-State Dynamic Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A mode-based steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is a linear perturbation procedure;
- calculates the response based on the system's eigenfrequencies and modes;
- requires that an eigenfrequency extraction procedure be performed prior to the steady-state dynamic analysis;
- can use the high-performance SIM software architecture (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*);
- can include nondiagonal damping effects (that is, from material or element damping) only when using the SIM architecture;
- is an alternative to direct-solution steady-state dynamic analysis, in which the system's response is calculated in terms of the physical degrees of freedom of the model;
- can include computation of acoustic contribution factors to help determine the major contributors to acoustic noise;
- is computationally cheaper than direct-solution or subspace-based steady-state dynamics;
- is less accurate than direct-solution or subspace-based steady-state analysis, in particular if significant material damping is present, and
- is able to bias the excitation frequencies toward the values that generate a response peak.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Dynamic Analysis Procedures*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Subspace-Based Steady-State Dynamic Analysis*
- *Output to the Output Database*
- **STEADY STATE DYNAMICS*
- *Configuring a mode-based steady-state dynamic analysis*

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system due to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep by applying the loading at a series of different frequencies and recording the response; in Abaqus/Standard the steady-state dynamic analysis procedure is used to conduct the frequency sweep.

In a mode-based steady-state dynamic analysis, the response is based on modal superposition techniques; the modes of the system must first be extracted using the eigenfrequency extraction procedure. The modes will

include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

When defining a mode-based steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below (*Selecting the Frequency Spacing*). Logarithmic frequency spacing is the default. Frequencies are given in cycles/time.

These frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (see *The Bias Parameter* below).

While the response in this procedure is for linear vibrations, the prior response can be nonlinear. Initial stress effects (stress stiffening) will be included in the steady-state dynamics response if nonlinear geometric effects (*General and Perturbation Procedures*) were included in any general analysis step prior to the eigenfrequency extraction step preceding the steady-state dynamic procedure.

The spectral solver can perform better than the conventional solver in the analyses with many modes, frequency points, and load cases for finite element models with special, but commonly used, damping properties (for example, no viscous damping in the model, viscous damping specified for relatively small portions of the model, or viscous modal damping specified for several selected modes).

Input File Usage: Use the following option to conduct a mode-based steady-state dynamic analysis:

**STEADY STATE DYNAMICS*

Use the following option to select the conventional mode-based solver:

**STEADY STATE DYNAMICS*, MODE BASED=CONVENTIONAL

Use the following option to select the spectral mode-based solver:

**STEADY STATE DYNAMICS*, MODE BASED=SPECTRAL

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**

Selecting the Type of Frequency Interval for Which Output Is Requested

Three types of frequency intervals are permitted for output from a mode-based steady-state dynamic step.

Specifying the Frequency Ranges by Using the System's Eigenfrequencies

By default, the eigenfrequency type of frequency interval is used; in this case the following intervals exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function (which is discussed below and allows the sampling points on the frequency scale to be spaced closer together at

eigenfrequencies in the frequency range). Thus, detailed definition of the response close to resonance frequencies is allowed. *Figure 1* illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

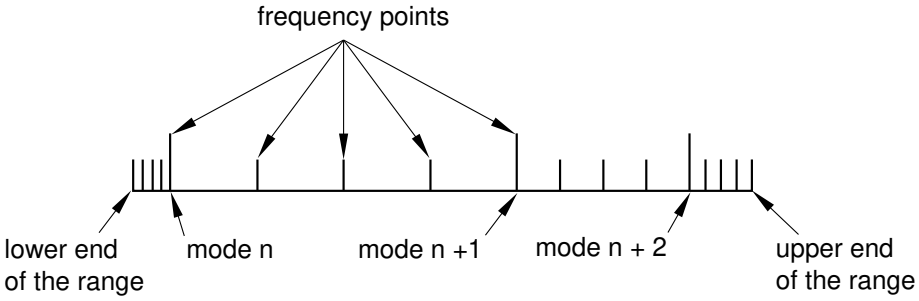


Figure 1: Division of range for the eigenfrequency type of interval and 5 calculation points.

Input File Usage: **STEADY STATE DYNAMICS*, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Use eigenfrequencies to subdivide each frequency range**

Specifying the Frequency Ranges by the Frequency Spread

If the spread type of frequency interval is selected, intervals exist around each eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. *Figure 2* illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

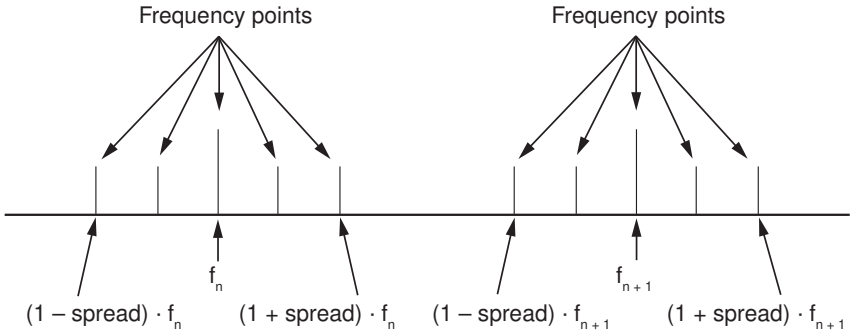


Figure 2: Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Input File Usage: **STEADY STATE DYNAMICS*, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

Specifying the Frequency Ranges Directly

If the alternative range type of frequency interval is chosen, there is only one interval in the specified frequency range spanning from the lower to the upper limit of the range. This interval is divided using the user-defined number of points and the optional bias function, which can be used to space the sampling frequency points closer to the range limits. For the range type of frequency interval, the peak responses around the system's eigenfrequencies might be missed since the sampling frequencies at which output will be reported will not be biased toward the eigenfrequencies.

Input File Usage: **STEADY STATE DYNAMICS*, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Selecting the Frequency Spacing

Two types of frequency spacing are permitted for a mode-based steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use either of the following options:

**STEADY STATE DYNAMICS*, FREQUENCY SCALE=LOGARITHMIC
**STEADY STATE DYNAMICS*, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Scale: Logarithmic** or **Linear**

Requesting Multiple Frequency Ranges

You can request multiple frequency ranges or multiple single frequency points for a mode-based steady-state dynamic step.

Input File Usage:
**STEADY STATE DYNAMICS*
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...

Repeat the data lines as often as necessary.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Data**: enter data in table, and add rows as necessary

The Bias Parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. *Figure 3* shows a few examples of the effect of the bias parameter on the frequency spacing.

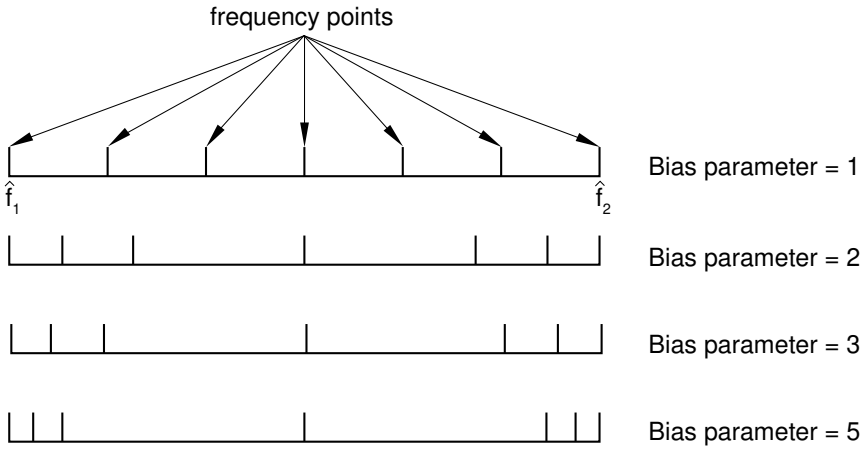


Figure 3: Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used to calculate the frequency at which results are presented is as follows:

$$\hat{f}_k = \tfrac{1}{2}\left(\hat{f}_1 + \hat{f}_2\right) + \tfrac{1}{2}\left(\hat{f}_2 - \hat{f}_1\right)|y|^{1/p}\,\mathrm{sign}\,(y)\,,$$

where

y

$= -1 + 2\,(k-1)\,/\,(n-1)$;

n

is the number of frequency points at which results are to be given within a frequency interval (discussed above);

k

is one such frequency point ($k = 1, 2, \ldots, n$);

$\hat{f}_1$

is the lower limit of the frequency interval;

$\hat{f}_2$

is the upper limit of the frequency interval;

$\hat{f}_k$

is the frequency at which the kth results are given;

p

is the bias parameter value; and

$\hat{f}$

is the frequency or the logarithm of the frequency, depending on the value used for the frequency scale parameter.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 3.0 for an eigenfrequency interval and 1.0 for a range frequency interval.

The Frequency Scale Factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see *Specifying the Frequency Ranges by Using the System's Eigenfrequencies* above).

Selecting the Modes and Specifying Damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the Modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS
**SELECT EIGENMODES*, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying Modal Damping

Damping is almost always specified for a steady-state analysis (see *Material Damping*). If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in *Material Damping*. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

**MODAL DAMPING*, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

**MODAL DAMPING*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**:
Steady-state dynamics, Modal: **Damping**

Example of Specifying Damping

Figure 4 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
f1, d1
f2, d2
f2, d3
f3, d3
f4, d4
```

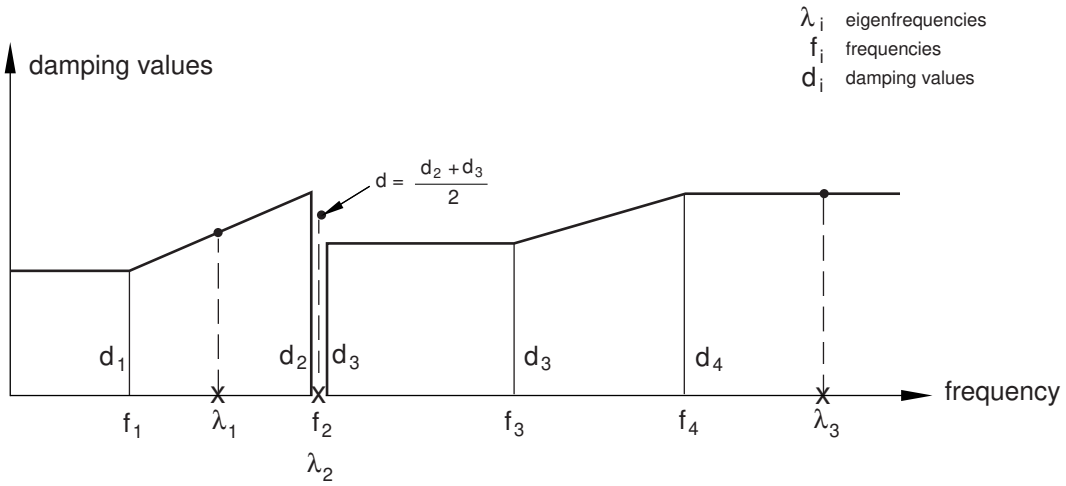


Figure 4: Damping values specified by frequency range.

Rules for Selecting Modes and Specifying Damping Coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Specifying Global Damping

For convenience you can specify constant global damping factors for all selected eigenmodes for mass and stiffness proportional viscous factors, as well as stiffness proportional structural damping. For further details, see *Damping in Dynamic Analysis*.

Input File Usage: Use the following option to define damping by global factors:

**GLOBAL DAMPING*, ALPHA=*factor*, BETA=*factor*, STRUCTURAL=*factor*

Abaqus/CAE Usage: Defining damping by global factors is not supported in Abaqus/CAE.

Material Damping

Structural and viscous material damping (see *Material Damping*) is taken into account in a SIM-based steady-state dynamic analysis. Since the projection of damping onto the mode shapes is performed only one time during the

frequency extraction step, significant performance advantages can be achieved by using the SIM-based steady-state dynamic procedure (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*).

If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

You can deactivate the structural or viscous damping in a mode-based steady-state dynamic procedure if desired.

Input File Usage: Use the following option to deactivate structural and viscous damping in a specific steady-state dynamic step:

**DAMPING CONTROLS*, STRUCTURAL=NONE, VISCOUS=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Initial Conditions

The base state is the current state of the model at the end of the last general analysis step before the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions (*Initial Conditions*). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary Conditions

In a mode-based steady-state dynamic analysis, both the real and imaginary parts of any degree of freedom are either restrained or unrestrained; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will automatically restrain both the real and imaginary parts of a degree of freedom even if only one part is restrained.

Base Motion

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions (*Boundary Conditions*) in mode-based dynamic response procedures. Therefore, in a mode-based steady-state dynamic analysis, the motion of nodes can be specified only as base motion; nonzero displacement or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. The method for prescribing base motion in modal superposition procedures is described in *Transient Modal Dynamic Analysis*.

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure, the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

When secondary bases are used, low frequency eigenmodes will be extracted for each “big” mass applied in the model. Use care when choosing the frequency lower limit range in such cases. The “big” mass modes are important in the modal superposition; however, the response at zero or arbitrarily low frequency level should not be requested since it forces Abaqus/Standard to calculate responses at frequencies between these “big” mass eigenfrequencies, which is not desirable.

Frequency-Dependent Base Motion

An amplitude definition can be used to specify the amplitude of a base motion as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**BASE MOTION*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *step_name*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Loads

The following loads can be prescribed in a mode-based steady-state dynamic analysis, as described in *Concentrated Loads*:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in *Abaqus Elements Guide*.

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Fluid flux loading cannot be used in a steady-state dynamic analysis.

Input File Usage: Use either of the following input lines to define the real (in-phase) part of the load:

**CLOAD or *DLOAD*
**CLOAD or *DLOAD*, REAL

Use the following input line to define the imaginary (out-of-phase) part of the load:

**CLOAD or *DLOAD*, IMAGINARY

Abaqus/CAE Usage: Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**

Frequency-Dependent Loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**CLOAD or *DLOAD*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:***name*

Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part***i**: **Amplitude:***name*

Predefined Fields

Predefined temperature fields are not allowed in mode-based steady-state dynamic analysis. Other predefined fields are ignored.

Material Options

As in any dynamic analysis procedure, mass or density (*Density*) must be assigned to some regions of any separate parts of the model where dynamic response is required. The following material properties are not active during mode-based steady-state dynamic analyses: plasticity and other inelastic effects, viscoelastic effects, thermal properties, mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see *General and Perturbation Procedures*.

Elements

Any of the following elements available in Abaqus/Standard can be used in a steady-state dynamics procedure:

- stress/displacement elements (other than generalized axisymmetric elements with twist);
- acoustic elements;
- piezoelectric elements; or
- hydrostatic fluid elements.

See *Choosing the Appropriate Element for an Analysis Type*.

Output

In mode-based steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output, the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see *Abaqus/Standard Output Variable Identifiers*). In this case the first printed line in the data file gives the magnitude while the second gives the phase angle.

In steady-state dynamic analysis procedures, you can request output for load cases to store only relevant results for each load case. This can reduce the size of the output database.

For more information, see *Variables Available for Mode-Based Steady-State Dynamic Analysis*.

Total Energy Output

The energy variables that can be written to the output database are defined in *Total Energy Output Quantities*. In mode-based and direct-solution steady-state dynamic analyses the following energy output variables are available: ALLWK, ALLKE, ALLKEA, ALLKEP, ALLSE, ALLSEA, ALLSEP, ALLVD, ALLVDE, ALLVDG, ALLVDM, ALLHD, ALLHDE, ALLHDG, ALLHDM, EFLOW, PFLOW, RADEN, RADPOW, and ALLERPWR.

The following energies are not available as element set quantities: ALLWK, ALLVDM, and ALLHDM.

Energy dissipation due to viscous and structural damping is represented by the following output variables: ALLVD, ALLVDE, ALLVDG, ALLVDM, ALLHD, ALLHDE, ALLHDG, and ALLHDM. In addition, you can examine energy loss due to material, global, and modal damping as represented by the following output variables: ALLVDE and ALLHDE for material damping, ALLVDG and ALLHDG for global damping, and ALLVDM and ALLHDM for modal damping.

Input File Usage: **ENERGY OUTPUT*
list of output variables

Abaqus/CAE Usage: Step module: history output request editor: **Select from list below**

Energy and Power Flow

Mode-based and direct-solution steady-state dynamic analyses support the computation of the energy and power flow from/into a portion of the model (represented by an element set that is comprised of structural elements) through a boundary (represented by a node set or a structural element-based surface). Energy flow is represented by output variable EFLOW, while power flow is represented by output variable PFLOW.

As shown in *Figure 5*, computation of the energy flow and power flow requires either (a) specification of a structural element set and a node set that represents the boundary of the element set through which the energy/power flows or (b) specification of a structural element set and an element-based surface through which the energy/power flows.

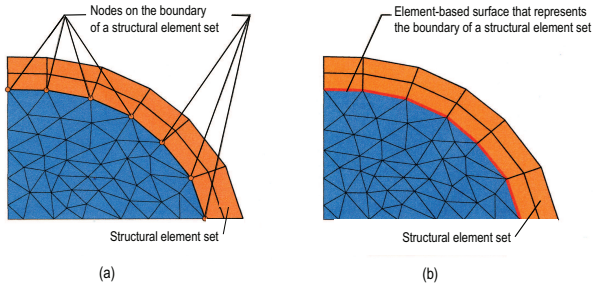


Figure 5: Requirements of computation of energy and power flow.

Input File Usage: **ENERGY OUTPUT*, ELSET=*structural_elset_name*, NSET=*structural_nset_name*
EFLOW, PFLOW

**ENERGY OUTPUT*, ELSET=*structural_elset_name*,
SURFACE=*structural_surface_name*
EFLOW, PFLOW

Abaqus/CAE Usage: Output for energy and power flow is not supported in Abaqus/CAE.

Radiated Energy and Power

Mode-based and direct-solution steady-state dynamic analyses support the computation of the radiated acoustic energy and power from/into an acoustic cavity (represented by an element set of acoustic elements) through a portion of the structural-acoustic interface (represented by a node set containing nodes of structural elements or by a structural element-based surface).

Radiated energy is represented by output variable RADEN, while radiated power is represented by output variable RADPOW. The element set representing the acoustic cavity can consist of a single element in that acoustic cavity. Abaqus/Standard computes the contribution of the other acoustic elements belonging to the same cavity automatically.

As shown in *Figure 6*, computation of the acoustic radiated energy and power requires the specification of an acoustic element set that represents the acoustic cavity; as well as either (a) a node set of the structural nodes on the boundary of the cavity or (b) an element-based surface that represents the structural boundary of the acoustic cavity or its portion.

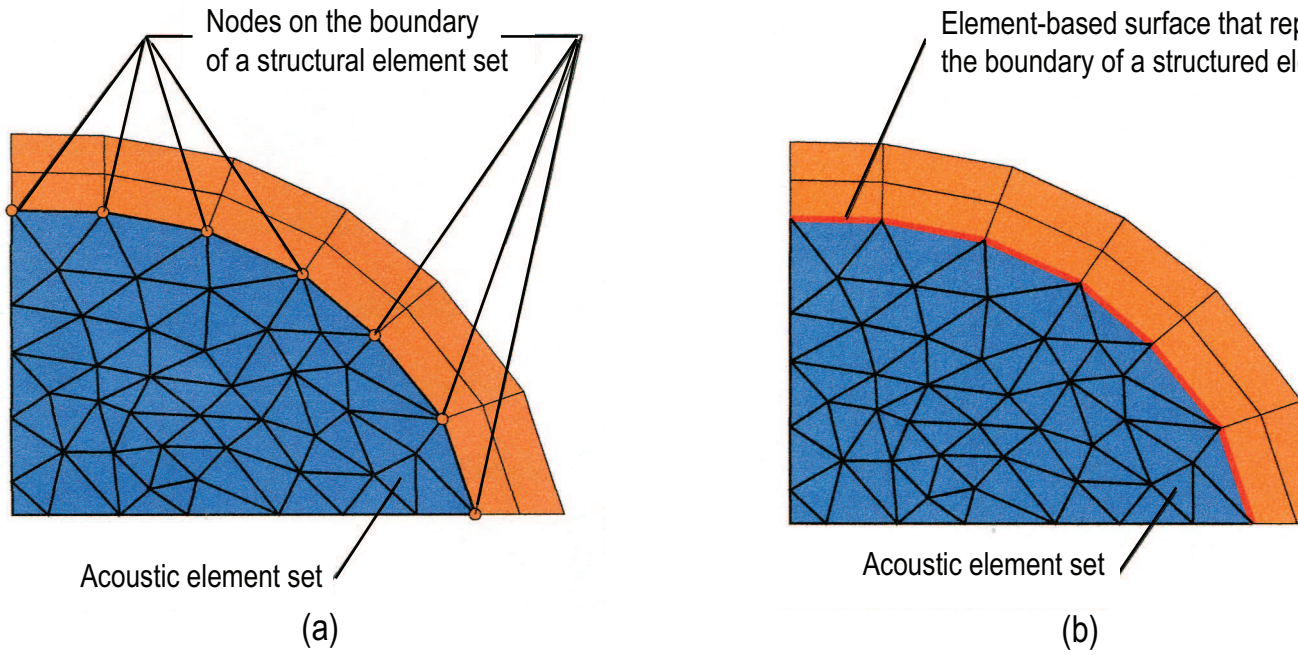


Figure 6: Requirements for computation of acoustic radiated energy and power.

Input File Usage: **ENERGY OUTPUT*, ELSET=*acoustic_elset_name*, NSET=*structural_nset_name*
RADEN, RADPOW

**ENERGY OUTPUT*, ELSET=*acoustic_elset_name*,
SURFACE=*structural_surface_name*
RADEN, RADPOW

Abaqus/CAE Usage: Output for radiated energy and power is not supported in Abaqus/CAE.

Equivalent Radiated Power

Computation of the equivalent radiated power can be required when you need to obtain the acoustic power emitted from structural elements, but either the model contains no acoustic elements or the structural surface has no contact with the acoustic elements. In such cases output variables ERPWR, ERPWRDEN, and ALLERPWR can be used to output the equivalent radiated power, its density, and the total radiated power. ERPWR represents the equivalent power that is radiated from the tributary area of a node on the structural element-based surface, while ERPWRDEN represents the density of the equivalent radiated power at a structural node. Variable ALLERPWR variable is used to calculate the equivalent radiated power emitted by a panel (represented by a structural element-based surface).

To output ERPWR, ERPWRDEN, and ALLERPWR, the radiating structural element-based surface with the acoustic medium properties specified must be used instead of the node and element sets.

If two or more surfaces have common points, Abaqus/Viewer will show those values of variables ERPWR/ERPWRDEN that were last written to the analysis database.

Input File Usage: **SURFACE PROPERTY*, NAME=*property_name*
**EQUIVALENT RADIATED SURFACE PROPERTIES*
density, *sound_speed*, *radiation_factor*
. . .

**NODE OUTPUT*, SURFACE=*structural_surface_name*
ERPWR, ERPWRDEN
**ENERGY OUTPUT*, SURFACE=*structural_surface_name*
ALLERPWR

Abaqus/CAE Usage: Output for equivalent radiated power is not supported in Abaqus/CAE.

Whole Element Energy Output

The whole element energy variables that can be written to the output database are defined in *Whole Element Energy Density Variables*. Modal steady-state dynamic analysis supports the computation of mean values of kinetic and potential energies in the finite elements (ELKE and ELSE) as well as the total energy loss for the period due to viscous and structural damping (ELVD, ELVDE, ELVDG, ELHD, ELHDE, and ELHDG).

Computation of the amplitude and peak values of the kinetic and potential energies is provided (ELKEA, ELKEP, ELSEA, and ELSEP). In addition, computation of various energy densities is supported (EKEDEN, EKEDENA, EKEDENP, ESEDEN, ESEDENA, ESEDENP, EVDDEN, EVDDENE, EVDDENG, EHDDEN, EHDDENE, and EHDDENG).

Acoustic Contribution Factors

Computation of the acoustic contribution factors helps you determine the major noise sources. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. For more information, see *Acoustic Contribution Factors in Mode-Based and Subspace-Based Steady-State Dynamic Analyses*.

Variables Available for Mode-Based Steady-State Dynamic Analysis

The following variables are provided specifically for steady-state dynamic analysis:

Element integration point variables:

PHS Magnitude and phase angle of all stress components.

PHE Magnitude and phase angle of all strain components.

PHEPG Magnitude and phase angles of the electrical potential gradient vector.

PHEFL Magnitude and phase angles of the electrical flux vector.

PHMFL Magnitude and phase angle of the mass flow rate in fluid link elements.

PHMFT Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF Magnitude and phase angle of connector total forces.

PHCEF Magnitude and phase angle of connector elastic forces.

PHCVF Magnitude and phase angle of connector viscous forces.

PHCRF Magnitude and phase angle of connector reaction forces.

PHCSF Magnitude and phase angle of connector friction forces.

PHCU Magnitude and phase angle of connector relative displacements.

PHCCU Magnitude and phase angle of connector constitutive displacements.

Nodal variables:

PU Magnitude and phase angle of all displacement/rotation components at a node.

PPOR Magnitude and phase angle of the fluid or acoustic pressure at a node.

PHPOT Magnitude and phase angle of the electrical potential at a node.

PRF Magnitude and phase angle of all reaction forces/moments at a node.

PHCHG Magnitude and phase angle of the reactive charge at a node.

The following energy output variables are available in both mode-based and direct-solution steady-state dynamic analyses (unless mentioned otherwise):

Total energy output variables:

ALLKE Kinetic energy. In steady-state dynamic analysis this is the cyclic mean value.

ALLKEA Kinetic energy amplitude.

ALLKEP Kinetic energy peak value.

ALLSE Recoverable strain energy. In steady-state dynamic analysis this is the cyclic mean value.

ALLSEA Recoverable strain energy amplitude.

ALLSEP Recoverable strain energy peak value.

ALLVD Energy dissipated by viscous effects including viscous regularization (except for cohesive elements and cohesive contact), not inclusive of energy dissipated by automatic stabilization and viscoelasticity. If this variable is requested for the whole model, it includes energy loss due to the material, global, and modal damping. If this variable is requested for a part of the model, energy loss due to the modal damping is not included.

ALLVDE Energy dissipated by viscous effects due to the material damping.

ALLVDG Energy dissipated by viscous effects due to the global damping.

ALLVDM Energy dissipated by viscous effects due to the modal damping. This variable is available for the whole model in a mode-based steady-state dynamic analysis only.

ALLHD Energy dissipated due to the structural damping. If this variable is requested for the whole model, it includes energy loss due to the material, global, and modal damping. If this variable is requested for a part of the model, energy loss due to the modal damping is not included.

ALLHDE Energy dissipated due to the material structural damping.

ALLHDG Energy dissipated due to the global structural damping.

ALLHDM Energy dissipated due to the modal structural damping. This variable is available for the whole model in a mode-based steady-state dynamic analysis only.

ALLWK External work. (Available only for the whole model.)

EFLOW Energy flow from a given portion of the model through the given boundary.

PFLOW Power flow from a given portion of the model through the given boundary.

RADEN Radiated energy from/into a given acoustic cavity through the given boundary.

RADPOW Radiated power from/into a given acoustic cavity through the given boundary.

ALLERPWR Total equivalent radiated power emitted by a panel that is represented by a structural element-based surface.

Nodal equivalent radiated power variables:

ERPWR Equivalent radiated power emitted from a tributary area of a structural element-based surface node.

ERPWRDEN Equivalent radiated power density emitted from a tributary area of a structural element-based surface node.

ERPAC Equivalent radiated acoustic pressure at a structural element-based surface node.

Whole element energy variables:

ELKE Total kinetic energy in the element. In steady-state dynamic analysis this is the cyclic mean value.

ELKEA Total kinetic energy amplitude in the element.

ELKEP Total kinetic energy peak value in the element.

ELSE Total elastic strain energy in the element. When the Mullins effect is modeled with hyperelastic materials, this quantity represents only the recoverable part of energy in the element. In steady-state dynamic analysis this is the cyclic mean value.

ELSEA Total elastic strain energy amplitude in the element.

ELSEP Total elastic strain energy peak value in the element.

ELVD Total energy dissipated in the element by viscous effects, not including energy dissipated by static stabilization or viscoelasticity.

ELVDE Total energy dissipated in the element by viscous effects due to the material damping.

ELVDG Total energy dissipated in the element by viscous effects due to the global damping.

ELHD Total energy dissipated in the element due to structural damping. This variable includes energy loss due to the material and global structural damping.

ELHDE Total energy dissipated in the element due to the material structural damping.

ELHDG Total energy dissipated in the element due to the global structural damping.

Whole element energy density variables:

EKEDEN Kinetic energy density in the element. In steady-state dynamic analysis this is the cyclic mean value.

EKEDENA Kinetic energy density amplitude in the element.

EKEDENP Kinetic energy density peak value in the element.

ESEDEN Total elastic strain energy density in the element. When the Mullins effect is modeled with hyperelastic materials, this quantity represents only the recoverable part of energy density in the element. This variable is not available in eigenvalue extraction procedures. In steady-state dynamic analysis this is the cyclic mean value.

ESEDENA Total elastic strain energy density amplitude in the element.

ESEDENP Total elastic strain energy density peak value in the element.

EVDDEN Total energy dissipated per unit volume in the element by viscous effects, not inclusive of energy dissipated through static stabilization or viscoelasticity.

EVDDENE Total energy dissipated per unit volume in the element by viscous effects due to the material damping.

EVDDENG Total energy dissipated per unit volume in the element by viscous effects due to the global damping.

EHDDEN Total energy dissipated per unit volume in the element due to structural damping. This variable includes energy loss due to the material and global structural damping.

EHDDENE Total energy dissipated per unit volume in the element due to the material structural damping.

EHDDENG Total energy dissipated per unit volume in the element due to the global structural damping.

The standard output variables U, V, A, and the variable PU listed above correspond to motions relative to the motion of the primary base in a mode-based analysis. Total values, which include the motion of the primary base, are also available:

TU Magnitude of all components of total displacement/rotation at a node.

TV Magnitude of all components of total velocity at a node.

TA Magnitude of all components of total acceleration at a node.

PTU Magnitude and phase angle of all total displacement/rotation components at a node.

The following modal variables are also available for mode-based steady-state dynamic analysis and can be output to the data, results, and/or output database files (see *Output to the Data and Results Files* and *Output to the Output Database*):

GU Generalized displacements for all modes.

GV Generalized velocities for all modes.

GA Generalized accelerations for all modes.

GPU Phase angle of generalized displacements for all modes.

GPV Phase angle of generalized velocities for all modes.

GPA Phase angle of generalized acceleration for all modes.

SNE Elastic strain energy for the entire model per mode.

KE Kinetic energy for the entire model per mode.

T External work for the entire model per mode.

BM Base motion.

Whole model variables such as ALLIE (total strain energy) are available for mode-based steady-state dynamics as output to the data, results, and/or output database files (see *Output to the Data and Results Files*).

Input File Template

```
*HEADING
…
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
*AMPLITUDE, NAME=base
Data lines to define an amplitude curve to be used to prescribe base motion
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamics step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
*END STEP
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*BOUNDARY, BASE NAME=base2
Data lines to assign degrees of freedom to a secondary base
*END STEP
**
*STEP
*STEADY STATE DYNAMICS
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*ACOUSTIC CONTRIBUTION
*MODAL DAMPING
Data lines to define the modal damping factors
*BASE MOTION, DOF=dof, AMPLITUDE=base
*BASE MOTION, DOF=dof, AMPLITUDE=base, BASE NAME=base2
*CLOAD and/or *DLOAD, AMPLITUDE=loadamp
Data lines to specify sinusoidally varying, frequency-dependent loads
…
*END STEP
```

Subspace-Based Steady-State Dynamic Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A subspace-based steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is based on projection of the steady-state dynamic equations on a subspace of selected modes of the undamped system;
- is a linear perturbation procedure;
- provides a cost-effective way to include frequency-dependent effects (such as frequency-dependent damping and viscoelastic effects) in the model;
- allows for nonsymmetric stiffness;
- requires that an eigenfrequency extraction procedure be performed prior to the steady-state dynamic analysis;
- can use the high-performance SIM software architecture (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*);
- is an alternative to direct-solution steady-state dynamic analysis, in which the system's response is calculated in terms of the physical degrees of freedom of the model;
- can include computation of acoustic contribution factors to help determine the major contributors to acoustic noise;
- is computationally cheaper than direct-solution steady-state dynamics but more expensive than mode-based steady-state dynamics;
- is less accurate than direct-solution steady-state analysis, in particular if significant material damping or viscoelasticity with a high loss modulus is present; and
- is able to bias the excitation frequencies toward the values that generate a response peak.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Dynamic Analysis Procedures*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Mode-Based Steady-State Dynamic Analysis*
- *Output to the Output Database*
- **STEADY STATE DYNAMICS*
- *Configuring a subspace-based steady-state dynamic analysis*

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system subjected to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep, by applying

the loading at a series of different frequencies and recording the response. In Abaqus/Standard the subspace-based steady-state dynamic analysis procedure is used to conduct the frequency sweep.

In a subspace-based steady-state dynamic analysis the response is based on direct solution of the steady-state dynamic equations projected onto a subspace of modes. The modes of the undamped, symmetric system must first be extracted using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The procedure is based on the assumption that the forced steady-state vibration can be represented accurately by a number of modes of the undamped system that are in the range of the excitation frequencies of interest. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part. The projection of the dynamic equilibrium equations onto a subspace of selected modes leads to a small system of complex equations that is solved for modal amplitudes, which are then used to compute nodal displacements, stresses, etc.

When defining a subspace-based steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below (*Selecting the Frequency Spacing*). Logarithmic frequency spacing is the default if the frequency ranges are specified directly or by eigenfrequencies. If the frequency ranges are specified by the frequency spread, only linear spacing can be used. Frequencies should be given in cycles/time.

The frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (see *The Bias Parameter* below).

The subspace-based steady-state dynamic analysis procedure can be used:

- for nonsymmetric stiffness;
- when any form of damping (except modal damping) is included; and
- when viscoelastic material properties must be taken into account.

While the response in this procedure is for linear vibrations, the prior response can be nonlinear. Initial stress effects (stress stiffening) will be included in the steady-state dynamic response if nonlinear geometric effects (*General and Perturbation Procedures*) were included in any general analysis step prior to the eigenfrequency extraction step preceding the subspace-based steady-state dynamic procedure.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**

Ignoring Damping

If damping terms can be ignored, you can specify that a real, rather than a complex, system matrix be generated and projected, which can significantly reduce computational time, at the cost of ignoring the damping effects.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, REAL ONLY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Compute real response only**

Selecting the Type of Frequency Interval for Which Output Is Requested

Three types of frequency intervals are permitted for output from a subspace-based steady-state dynamic step.

Specifying the Frequency Ranges by Using the System's Eigenfrequencies

By default, the eigenfrequency type of frequency interval is used; in this case the following intervals exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function (which is discussed below and allows the sampling points on the frequency scale to be spaced closer together at eigenfrequencies in the frequency range). Thus, detailed definition of the response close to resonance frequencies is allowed. *Figure 1* illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

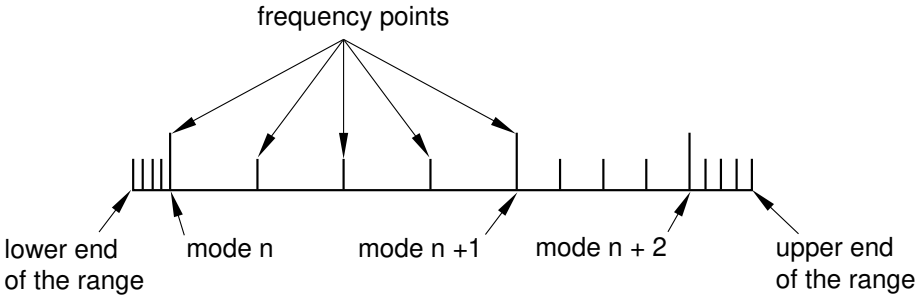


Figure 1: Division of range for the eigenfrequency type of interval and 5 calculation points.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Use eigenfrequencies to subdivide each frequency range**

Specifying the Frequency Ranges by the Frequency Spread

If the spread type of frequency interval is selected, intervals exist around each eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. *Figure 2* illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

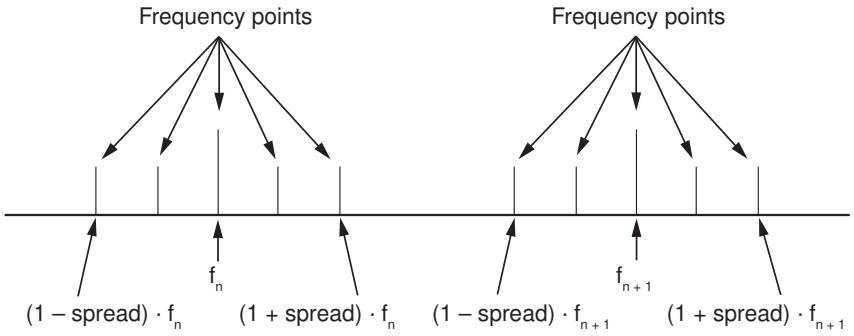


Figure 2: Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

Specifying the Frequency Ranges Directly

If the alternative range type of frequency interval is chosen, there is only one interval in the specified frequency range spanning from the lower to the upper limit of the range. This interval is divided using the user-defined number of points and the optional bias function, which can be used to space the sampling frequency points closer to the range limits. For the range type of frequency interval, the peak responses around the system's eigenfrequencies may be missed since the sampling frequencies at which output will be reported will not be biased toward the eigenfrequencies.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Selecting the Frequency Spacing

Two types of frequency spacing are permitted for a subspace-based steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use the following option to specify logarithmic frequency spacing:

**STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, FREQUENCY SCALE=LOGARITHMIC (default)

Use the following option to specify linear frequency spacing:

**STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Scale: Logarithmic** or **Linear**

Requesting Multiple Frequency Ranges

You can request multiple frequency ranges for a subspace-based steady-state dynamic step. When both frequency ranges and additional single frequency points are requested, the frequency ranges must be specified first.

Input File Usage: Repeat the data lines as often as necessary to request multiple frequency ranges or multiple single frequency points:

**STEADY STATE DYNAMICS*, SUBSPACE PROJECTION
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Data**: enter data in table, and add rows as necessary

The Bias Parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. *Figure 3* shows a few examples of the effect of the bias parameter on the frequency spacing.

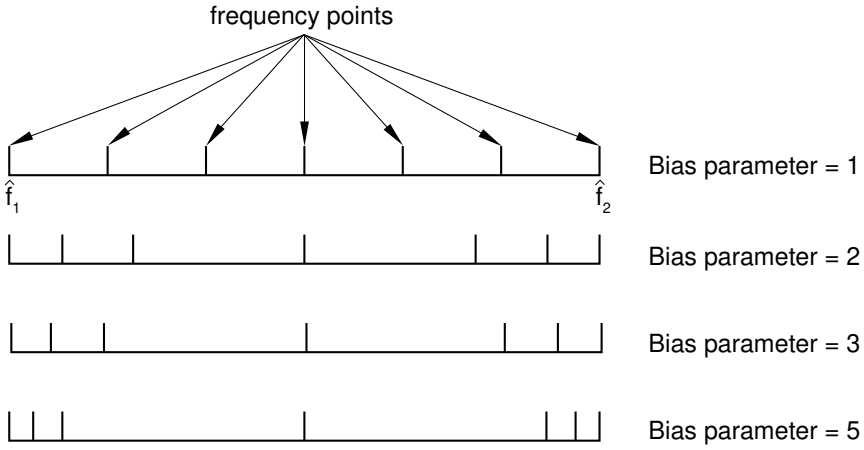


Figure 3: Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used in subspace-based steady-state dynamics is

$$\hat{f}_k = \frac{1}{2}\left(\hat{f}_1 + \hat{f}_2\right) + \frac{1}{2}\left(\hat{f}_2 - \hat{f}_1\right)|y|^{1/p}\,\mathrm{sign}\,(y)\,,$$

where

y

$= -1 + 2(k-1)/(n-1)$;

n

is the number of frequency points at which results are to be given within a frequency interval (discussed above);

k

is one such frequency point ($k = 1, 2, \ldots, n$);

$\hat{f}_1$

is the lower limit of the frequency interval;

$\hat{f}_2$

is the upper limit of the frequency interval;

$\hat{f}_k$

is the frequency at which the kth results are given;

p

is the bias parameter value; and

$\hat{f}$

is the frequency or the logarithm of the frequency, depending on the value chosen for the frequency scale.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 3.0 for an eigenfrequency interval and 1.0 for a range frequency interval.

The Frequency Scale Factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see *Specifying the Frequency Ranges by Using the System's Eigenfrequencies* above).

Damping

If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate

specification of damping properties is essential. The various damping options available are discussed in *Material Damping*.

In subspace-based steady-state dynamic analysis damping can be created by the following:

- dashpots (see *Dashpots*),
- "Rayleigh" damping associated with materials and elements (see *Material Damping*),
- structural damping (see *Damping in Dynamic Analysis*),
- viscoelasticity included in the material definitions (see *Frequency Domain Viscoelasticity*),
- contributions from infinite elements (see *Infinite Elements*) or defined impedance conditions (see *Acoustic and Shock Loads*) on acoustic elements, and
- "volumetric drag" (viscous Rayleigh damping) in acoustic elements (see *Acoustic Medium*).

If you specify that a real-only system matrix be generated and projected (see *Ignoring Damping* above), all forms of damping are ignored, including quiet boundaries on infinite elements and nonreflecting boundaries on acoustic elements.

Contact Conditions with Sliding Friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom are not constrained and the effect of friction results in an unsymmetric contribution to the stiffness matrix. At other contact nodes the tangential degrees of freedom are constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see *Coulomb friction*. Subspace-based steady-state dynamics analysis allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Include friction-induced damping effects**

Selecting the Modes on Which to Project

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use the following option to select the modes by specifying mode numbers individually:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS

Use the following option to request that Abaqus/Standard generate the mode numbers automatically:

*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Selecting the Subspace Projection Frequency

You can control the frequency of the subspace projections. By default, the dynamic equations are projected onto the subspace at each frequency you request. However, considerable computational savings can be obtained if the projection onto the subspace is performed only at selected frequency points.

Projecting the Subspace at Each Frequency Requested

By default, the dynamic equations are projected onto the subspace at each frequency you requested. This is the most computationally expensive method.

Input File Usage: Use either of the following options:

*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
*STEADY STATE DYNAMICS,
SUBSPACE PROJECTION=ALL FREQUENCIES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Evaluate at each frequency**

Projecting the Subspace Using Model Properties at the Center Frequency of All Ranges

You can perform only one projection using model properties evaluated at the center frequency of all ranges and individual frequency points specified. The center frequency is determined on a logarithmic or linear scale depending on the spacing requested.

This method is the least expensive. However, it should be chosen only when the material properties do not depend strongly on frequency.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION=CONSTANT

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Constant**

Projecting the Subspace at Each Extracted Eigenfrequency

You can perform the projections at each extracted eigenfrequency in the requested frequency range and at eigenfrequencies immediately outside the range. The projected mass, stiffness, and damping matrices are then interpolated at each frequency point requested. The interpolation is performed on a linear or logarithmic scale depending on the spacing requested.

Input File Usage: *STEADY STATE DYNAMICS,
SUBSPACE PROJECTION=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Interpolate at eigenfrequencies**

Projecting the Subspace Based on Material Property Changes as a Function of Frequency

You can select how often subspace projections are performed based on material property changes as a function of frequency. You specify the relative change in material stiffness and damping properties allowed before a new projection is performed. In the beginning of the subspace-based steady-state dynamic step Abaqus/Standard computes a table of relative changes in material stiffness and damping properties, and projections are performed based on the strictest of the two criteria. The projections are then interpolated at each requested frequency point as described above. The default value for the allowable stiffness or damping change is 0.1.

Input File Usage: **STEADY STATE DYNAMICS*,
SUBSPACE PROJECTION=PROPERTY CHANGE,
DAMPING CHANGE=*percentage*, STIFFNESS CHANGE=*percentage*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **As a function of property changes**, **Max. damping change**: *percentage*, **Max. stiffness change**: *percentage*

Projecting the Subspace at the Limits of Each Frequency Range

You can select how often subspace projections are performed based on the limits of each frequency range. The projections onto the modal subspace of the dynamic equations are performed at the lower limit of each frequency range and at the upper limit of the last frequency range. The interpolation of the projected mass, stiffness, and damping matrices is performed on a linear scale. This method can be used only with the SIM architecture.

This method should be chosen when the frequency dependence of material properties is close to linear within a frequency range.

Input File Usage: **STEADY STATE DYNAMICS*, SUBSPACE PROJECTION=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Interpolate at lower and upper frequency limits**

Initial Conditions

The base state is the current state of the model at the end of the last general analysis step prior to the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions (*Initial Conditions*). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary Conditions

In a subspace-based steady-state dynamic analysis both the real and imaginary parts of any degree of freedom are either restrained or unrestrained; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will restrain both the real and imaginary parts of a degree of freedom automatically even if only one part is restrained.

Base Motion

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions (*Boundary Conditions*) in subspace-based steady-state dynamic analysis. Instead, prescribed motion can be specified as base motion; nonzero displacement or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. The method for prescribing base motion in modal superposition procedures is described in *Transient Modal Dynamic Analysis*.

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

When secondary bases are used, low frequency eigenmodes are extracted for each “big” mass applied in the model. Use care when choosing the lower limit range for the frequency in such cases. The “big” mass modes are important in the modal superposition. However, you should not request the response at zero or an arbitrarily low frequency level because this forces Abaqus/Standard to calculate the responses at frequencies between these “big” mass eigenfrequencies, which is not desirable.

Frequency-Dependent Base Motion

An amplitude definition can be used to specify the amplitude of a base motion as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**BASE MOTION*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *step_name*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Loads

The following loads can be prescribed in a subspace-based steady-state dynamic analysis, as described in *Concentrated Loads*:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in *Abaqus Elements Guide*.
- Incident wave loads can be applied; see *Acoustic and Shock Loads*. Incident wave loads can be used to model sound waves from distinct planar or spherical sources or from diffuse fields.

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Input File Usage: Use either of the following input lines to define the real (in-phase) part of the load:

**CLOAD or *DLOAD*
**CLOAD or *DLOAD*, REAL

Use the following input line to define the imaginary (out-of-phase) part of the load:

**CLOAD or *DLOAD*, IMAGINARY

Abaqus/CAE Usage: You can only define the real (in phase) part of the load in Abaqus/CAE.

Load module: load editor: *real (in-phase) part*

Frequency-Dependent Loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**CLOAD or *DLOAD*, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:***name*

Load module: load editor: **Amplitude:***name*

Loading Limitations

Coriolis distributed loading adds an imaginary antisymmetric contribution to the overall system of equations. This contribution is currently accounted for in solid and truss elements only and is activated by requesting the unsymmetric matrix storage and solution scheme for the step.

Fluid flux loading cannot be used in subspace-based steady-state dynamic analysis.

Predefined Fields

Predefined temperature fields can be specified in subspace-based steady-state dynamic analysis (see *Predefined Fields*) and can produce harmonically varying thermal strains if thermal expansion is included in the material definition (see *Computing Thermal Strains in Linear Perturbation Steps*). Other predefined fields are ignored.

Material Options

As in any dynamic analysis procedure, mass or density (*Density*) must be assigned to some regions of any separate parts of the model where dynamic response is required. If an analysis is desired in which the inertia effects are neglected, the density should be set to a very small number. Natural damping, as well as individual dashpots, can be included in this procedure.

Viscoelastic effects can be included in subspace-based steady-state dynamic analysis. The linearized viscoelastic response is considered to be a perturbation about a nonlinear preloaded state, which is computed on the basis of purely elastic behavior (long-term response) in the viscoelastic components. Therefore, the vibration amplitude must be sufficiently small so that the material response in the dynamic phase of the problem can be treated as a linear perturbation about the predeformed state. Viscoelastic frequency domain response is described in *Frequency Domain Viscoelasticity*.

The following material properties are not active during subspace-based steady-state dynamic analyses: plasticity and other inelastic effects, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see *General and Perturbation Procedures*.

Numerical investigations show that in general the accuracy of the results in the subspace-based steady-state dynamic step is improved if in the previous eigenfrequency extraction step the material properties are evaluated at a frequency in the vicinity of the center of the range spanned by the frequencies specified for the steady-state dynamic step (see *Natural Frequency Extraction*). In this case the modes extracted in the previous eigenfrequency extraction step for the undamped system will reflect most accurately the modes of the damped system at frequencies located in the proximity of the frequency at which the material properties are evaluated. Thus, if the steady-state dynamic response is sought for a large span of frequencies and the specified material properties vary significantly over this span, the results will be more accurate if the range is divided into smaller ranges and several separate analyses are run over these smaller ranges with the material properties evaluated at appropriate frequencies.

Elements

Any of the following elements available in Abaqus/Standard can be used in a subspace-based steady-state dynamic analysis:

- stress/displacement elements (other than generalized axisymmetric elements with twist);
- acoustic elements;
- piezoelectric elements; and
- hydrostatic fluid elements.

See *Choosing the Appropriate Element for an Analysis Type*.

Output

In subspace-based steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see *Abaqus/Standard Output Variable Identifiers*). In this case the first printed line in the data file gives the magnitude while the second gives the phase angle.

In steady-state dynamic analysis procedures, you can request output for load cases to store only relevant results for each load case. This can reduce the size of the output database.

The following variables are provided specifically for subspace-based steady-state dynamic analysis:

Element integration point variables:

PHS Magnitude and phase angle of all stress components.

PHE Magnitude and phase angle of all strain components.

PHEPG Magnitude and phase angles of the electrical potential gradient vector.

PHEFL Magnitude and phase angles of the electrical flux vector.

PHMFL Magnitude and phase angle of the mass flow rate in fluid link elements.

PHMFT Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF Magnitude and phase angle of connector total forces.

PHCEF Magnitude and phase angle of connector elastic forces.

PHCVF Magnitude and phase angle of connector viscous forces.

PHCRF Magnitude and phase angle of connector reaction forces.

PHCSF Magnitude and phase angle of connector friction forces.

PHCU Magnitude and phase angle of connector relative displacements.

PHCCU Magnitude and phase angle of connector constitutive displacements.

PHCV Magnitude and phase angle of connector relative velocities.

PHCA Magnitude and phase angle of connector relative accelerations.

Nodal variables:

PU Magnitude and phase angle of all displacement/rotation components at a node.

PPOR Magnitude and phase angle of the fluid or acoustic pressure at a node.

PHPOT Magnitude and phase angle of the electrical potential at a node.

PRF Magnitude and phase angle of all reaction forces/moments at a node.

PHCHG Magnitude and phase angle of the reactive charge at a node.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in a SIM-based, subspace-based steady-state dynamic analysis.

The standard output variables U, V, A, and the variable PU listed above correspond to motions relative to the motion of the primary base in a subspace-based steady-state dynamic analysis. Total values, which include the motion of the primary base, are also available:

TU Components of total displacement/rotation at a node.

TV Components of total velocity at a node.

TA Components of total acceleration at a node.

PTU Magnitude and phase angle of all total displacement/rotation components at a node.

The specified base motion is available for subspace-based steady-state dynamic analysis and can be output to the data, results, and/or output database files (see *Output to the Data and Results Files* and *Output to the Output Database*).

BM
Base motion.

Whole model variables such as ALLIE (total strain energy) are available for subspace-based steady-state dynamic analysis as output to the data, results, and/or output database files (see *Output to the Data and Results Files* and *Output to the Output Database*).

Acoustic Contribution Factors

Computation of the acoustic contribution factors helps you determine the major noise sources. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. For more information, see *Acoustic Contribution Factors in Mode-Based and Subspace-Based Steady-State Dynamic Analyses*.

Input File Template

```
*HEADING
…
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
*AMPLITUDE, NAME=base
Data lines to define an amplitude curve to be used to prescribe base motion
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamics step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
*END STEP
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*BOUNDARY, BASE NAME=base2
Data lines to assign degrees of freedom to a secondary base
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*BASE MOTION, DOF=dof, AMPLITUDE=base
*BASE MOTION, DOF=dof, AMPLITUDE=base, BASE NAME=base2
*CLOAD and/or *DLOAD, AMPLITUDE=loadamp
```

```
Data lines to specify sinusoidally varying, frequency-dependent loads
…
*END STEP
```

Response Spectrum Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A response spectrum analysis:

- provides an estimate of the peak linear response of a structure to dynamic motion provided in the form of a displacement, velocity, or acceleration spectrum;
- is typically used to analyze response to a seismic event;
- assumes that the system's response is linear so that it can be analyzed in the frequency domain using its natural modes, which must be extracted in a previous eigenfrequency extraction step (*Natural Frequency Extraction*);
- can use the high-performance SIM software architecture (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses*); and
- is a linear perturbation procedure and is, therefore, not appropriate if the excitation is so severe that nonlinear effects in the system are important.

References:

- *About Dynamic Analysis Procedures*
- *Defining an Analysis*
- *General and Perturbation Procedures*
- **RESPONSE SPECTRUM*
- **SPECTRUM*
- *Configuring a response spectrum procedure*
- *Defining a spectrum*

Response Spectrum Analysis

Response spectrum analysis can be used to estimate the peak response (displacement, stress, etc.) of a structure to a particular base motion or force. The method is only approximate, but it is often a useful, inexpensive method for preliminary design studies.

The response spectrum procedure is based on using a subset of the modes of the system, which must first be extracted by using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

If the number of eigenmodes included in the superposition does not sufficiently represent the total mass of the structure, you can use the missing mass method to augment the missing inertia in the dynamic response as described in *Using the Missing Mass Method*.

In cases with repeated eigenvalues and eigenvectors, the modal summation results must be interpreted with care. You should use mode combination rules that are appropriate for closely spaced modes.

While the response in the response spectrum procedure is for linear vibrations, the prior response may be nonlinear. Initial stress effects (stress stiffening) will be included in the response spectrum analysis if nonlinear geometric

effects (*General and Perturbation Procedures*) were included in a general analysis step prior to the eigenfrequency extraction step.

The problem to be solved can be stated as follows: given a set of base motions, $\ddot{u}_j^B(t)$ ($j = 1, 2, 3$), specified in orthogonal directions defined by direction cosines t_k^j ($k = 1, 2, 3$), estimate the peak value over all time of the response of any variable in a finite element model that is simultaneously subjected to these multiple base motions. The peak response is first computed independently for each direction of excitation for each natural mode of the system as a function of frequency and damping. These independent responses are then combined to create an estimate of the actual peak response of any variable chosen for output, as a function of frequency and damping.

The acceleration history (base motion) is not given directly in a response spectrum analysis; it must first be converted into a spectrum.

Specifying a Spectrum

The response spectrum method is based on first finding the peak response to each base motion excitation of a one degree of freedom system that has a natural frequency equal to the frequency of interest. The single degree of freedom system is characterized by its undamped natural frequency, ω_α, and the fraction of critical damping present in the system, ξ_α, at each mode α. The equations of motion of the system are integrated through time to find peak values of relative displacement, relative velocity, and relative or absolute acceleration for the linear, one degree of freedom system. This process is repeated for all frequency and damping values in the range of interest. Plots of these responses are known as displacement, velocity, and acceleration spectra: $S^D(\omega_\alpha, \xi_\alpha)$, $S^V(\omega_\alpha, \xi_\alpha)$, and $S^A(\omega_\alpha, \xi_\alpha)$. The response spectrum can be obtained directly from measured data, as described in *Defining a Spectrum Using Values of S as a Function of Frequency and Damping* below. You can also use a Fortran program to define a spectrum; an example of defining a spectrum from an acceleration record in this way is provided in *Analysis of a cantilever subject to earthquake motion*.

Alternatively, you can create the required spectrum by specifying an amplitude (time history record), the frequency range, and the damping values for which the spectrum will be built, as described in *Creating a Spectrum from a Given Time History Record* below. The spectrum can be used in the subsequent response spectrum analysis, or it can be written to a file for future use.

For each damping value the magnitude of the response spectrum must be given over the entire range of frequencies needed, in ascending value of frequency. Abaqus/Standard interpolates linearly between the values given on a log-log scale. Outside the extremes of the frequency range given, the magnitude is assumed to be constant, corresponding to the end value given. (See *Material Data Definition* for an explanation of data interpolation.)

Any number of spectra can be defined, and each spectrum must be named. The response spectrum procedure allows up to three spectra to be applied simultaneously to the model in orthogonal physical directions defined by their direction cosines.

Defining a Spectrum Using Values of *S* as a Function of Frequency and Damping

You can define a spectrum by specifying values for the magnitude of the spectrum; frequency, in cycles per time, at which the magnitude is used; and associated damping, given as a ratio of critical damping.

Input File Usage: To define the spectrum on the data lines:

**SPECTRUM*, NAME=*spectrum name*

Repeat this option to define multiple spectra for an analysis.

Abaqus/CAE Usage: To define a spectrum, do the following:

Step, Interaction, or Load module: **Tools**->**Amplitude**->**Create**; **Name**: *spectrum name*, **Type**: **Spectrum**

To apply a spectrum to the model, do the following:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Use response spectrum**: select spectrum name for each physical direction in which it should be applied

Specifying the Type of Spectrum

You can indicate whether a displacement, velocity, or acceleration spectrum is given. The default is an acceleration spectrum.

Alternatively, an acceleration spectrum can be given in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to define a displacement, velocity, or acceleration spectrum:

**SPECTRUM*, NAME=*name*, TYPE=DISPLACEMENT
**SPECTRUM*, NAME=*name*, TYPE=VELOCITY
**SPECTRUM*, NAME=*name*, TYPE=ACCELERATION

Use the following option to define an acceleration spectrum given in *g*-units:

**SPECTRUM*, NAME=*name*, TYPE=G, G=*g*

Abaqus/CAE Usage: Use one of the following options to define a displacement, velocity, or acceleration spectrum:

Step, Interaction, or Load module: **Tools**->**Amplitude**->**Create**; **Type**: **Spectrum**; **Specification units**: **Displacement**, **Velocity**, or **Acceleration**

Use the following option to define an acceleration spectrum given in *g*-units:

Step, Interaction, or Load module: **Tools**->**Amplitude**->**Create**; **Type**: **Spectrum**; **Specification units**: **Gravity**, **Gravity**: *g*

Reading the Data Defining the Spectrum from an Alternate Input File

The data for the spectrum can be specified in an alternate input file and read into the Abaqus/Standard input file.

Input File Usage: **SPECTRUM*, NAME=*name*, INPUT=*file name*

Abaqus/CAE Usage: Step, Interaction, or Load module: **Tools**->**Amplitude**->**Create**; **Type**: **Spectrum**; click mouse button 3 while holding the cursor over the data table, and select **Read from File**

Creating a Spectrum from a Given Time History Record

If you have a time history of a dynamic event (e.g., acceleration, velocity, displacement), you can build your own spectrum by specifying the record type and the amplitude name that this record represents. If the amplitude

record is given with an arbitrarily changing time increment, linear interpolation will be needed for the implicit integration scheme for the dynamic equation of motion for a single degree of freedom system subjected to this record. You can specify the frequency range for the integration scheme and the frequency increment. You can build a spectrum for every fraction of critical damping indicated in the list of damping values.

Input File Usage: **SPECTRUM*, CREATE, AMPLITUDE=*amplitude name*,
NAME=*spectrum name*, TIME INCREMENT=*dt*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Specifying the Type of Spectrum to Be Created

You can indicate whether a displacement, velocity, or acceleration spectrum is to be created. The default is an acceleration spectrum.

Alternatively, an acceleration spectrum can be created in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to create a displacement, velocity, or acceleration spectrum:

**SPECTRUM*, CREATE, TYPE=DISPLACEMENT
**SPECTRUM*, CREATE, TYPE=VELOCITY
**SPECTRUM*, CREATE, TYPE=ACCELERATION

Use the following option to create an acceleration spectrum in *g*-units:

**SPECTRUM*, CREATE, TYPE=G, G=*g*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Specifying the Record Type That the Time History Represents

You can indicate whether a displacement, velocity, or acceleration amplitude is specified. The default is an acceleration amplitude.

Alternatively, an acceleration amplitude can be given in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to indicate that the amplitude is defined in displacement, velocity, or acceleration units:

**SPECTRUM*, CREATE, EVENT TYPE=DISPLACEMENT
**SPECTRUM*, CREATE, EVENT TYPE=VELOCITY
**SPECTRUM*, CREATE, EVENT TYPE=ACCELERATION

Use the following option to indicate that an acceleration amplitude is given in *g*-units:

**SPECTRUM*, CREATE, EVENT TYPE=G, G=*g*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Creating an Absolute or Relative Acceleration Spectrum

When you create an acceleration spectrum from a given time history record, you can create an absolute or relative response spectrum. The default is an absolute spectrum.

Input File Usage: **SPECTRUM*, CREATE, TYPE=ACCELERATION, ABSOLUTE
**SPECTRUM*, CREATE, TYPE=ACCELERATION, RELATIVE

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Generating the List of Damping Values for the Fraction of Critical Damping

You must provide a list of damping values for the fraction of critical damping to create a spectrum. However, if the damping is evenly spaced between its lower and upper bound, you can automatically generate the list of damping values by providing the start value, end value, and increment for the fraction of critical damping.

Input File Usage: **SPECTRUM*, CREATE, DAMPING GENERATE

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Writing the Generated Spectra to an Independent File

You can write the generated spectra to an independent file. Otherwise, the generated spectra can be used only within the currently submitted job in subsequent response spectra procedures. You can inspect the generated spectra if you request that model definition data be printed to the data file (see *Model and History Definition Summaries*).

Input File Usage: **SPECTRUM*, CREATE, FILE=*file name*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Estimating the Peak Values of the Modal Responses

Since the response spectrum procedure uses modal methods to define a model's response, the value of any physical variable is defined from the amplitudes of the modal responses (the "generalized coordinates"), q_α. The first stage in the response spectrum procedure is to estimate the peak values of these modal responses. For mode α and spectrum k this is

$$(q_\alpha^{max})_k = c_k S_k^D(\omega_\alpha, \xi_\alpha) \sum_j t_j^k \Gamma_{\alpha j},$$

where

q_α

is the modal amplitude for mode α;

c_k

is a scaling parameter introduced as part of the response spectrum procedure definition for spectrum

$S_k^D(\omega_\alpha, \xi_\alpha)$;

$S_k^D(\omega_\alpha, \xi_\alpha)$

is the user-defined value of the spectrum (see *Specifying a Spectrum*) in direction k interpolated, if necessary, at natural frequency ω_α and the fraction of critical damping ξ_α in mode α;

t_j^k

is the jth direction cosine for the kth spectrum; and

$\Gamma_{\alpha j}$

is the participation factor for mode α in direction j (see *Natural Frequency Extraction*).

Similar expressions for $(\dot{q}_\alpha^{max})_k$ and $(\ddot{q}_\alpha^{max})_k$ can be obtained by substituting velocity or acceleration spectra in the above equation.

Combining the Individual Peak Responses

The individual peak responses to the excitations in different directions will occur at different times and, therefore, must be combined into an overall peak response. Two combinations must be performed, and both introduce approximations into the results:

1. The multidirectional excitations must be combined into one overall response. This combination is controlled by the directional summation method, as described below in *Directional Summation Methods*.
2. The peak modal responses must be combined to estimate the peak physical response. This combination is controlled by the modal summation method, as described below in *Modal Summation Methods*.

Depending on the type of base excitation, either modal responses or directional responses are combined first.

Directional Summation Methods

You choose the method for combining the multidirectional excitations depending on the nature of the excitations.

The Algebraic Method

If the input spectra in the different directions are components of a base excitation that is approximately in a single direction in space, then for each mode the peak responses in the different spatial directions are summed algebraically by

$$q_\alpha^{max} = \sum_k (q_\alpha^{max})_k.$$

After this summation is performed, the modal responses are summed. (Choosing the method used for modal summation is described below in *Modal Summation Methods*.) Since the directional components are summed first, the subscript k is not relevant and can be ignored in the modal summation equations that follow.

Input File Usage: **RESPONSE SPECTRUM*, COMP=ALGEBRAIC, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Single direction** or **Multiple direction absolute sum**

The Square Root of the Sum of the Squares Directional Summation Method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in *Modal Summation Methods*. Then, the estimates of the total peak response in each excitation direction $\left(R^i\right)_k^{max}$ are combined by

$$\left(R^i\right)^{max} = \sqrt{\sum_k \left(\left(R^i\right)_k^{max}\right)^2}.$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=SRSS, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction square root of the sum of squares**

The Forty-Percent Method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in *Modal Summation Methods*. Then, the responses in different excitation directions are combined by the 40% rule recommended by the ASCE 4–98 standard for Seismic Analysis of Safety-Related Nuclear Structures and Commentary, Section 3.2.7.1.2. This method combines the response for all possible combinations of the three components, including variations in sign (plus/minus), assuming that when the maximum response from one component occurs, the response from the other two components is 40% of their maximum value, using one of the following:

$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_1^{max} \pm 0.4\left(R^i\right)_2^{max} \pm 0.4\left(R^i\right)_3^{max}\right],$$
$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_2^{max} \pm 0.4\left(R^i\right)_1^{max} \pm 0.4\left(R^i\right)_3^{max}\right],$$
$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_3^{max} \pm 0.4\left(R^i\right)_2^{max} \pm 0.4\left(R^i\right)_1^{max}\right].$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=R40, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction forty percent rule**

The Thirty-Percent Method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in *Modal Summation Methods*. Then, the responses in different excitation directions are combined by the 30% rule recommended by the ASCE 4–98 standard for Seismic Analysis of Safety-Related Nuclear Structures and Commentary, Section 3.2.7.1.2. This method combines the response for all possible combinations of the three components, including variations in sign (plus/minus), assuming that when the maximum response from one component occurs, the response from the other two components is 30% of their maximum value, using one of the following:

$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_1^{max} \pm 0.3\left(R^i\right)_2^{max} \pm 0.3\left(R^i\right)_3^{max}\right],$$
$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_2^{max} \pm 0.3\left(R^i\right)_1^{max} \pm 0.3\left(R^i\right)_3^{max}\right],$$
$$\left(R^i\right)^{max} = \pm\left[\left(R^i\right)_3^{max} \pm 0.3\left(R^i\right)_2^{max} \pm 0.3\left(R^i\right)_1^{max}\right].$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=R30, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction thirty percent rule**

Modal Summation Methods

The peak response of some physical variable R^i (a component i of displacement, stress, section force, reaction force, etc.) caused by the motion in the αth natural mode excited by the given response spectra in direction k at frequency ω_α with damping ξ_α is given by

$$\left(R_\alpha^i\right)_k^{max} = \Phi_\alpha^i \left(q_\alpha^{max}\right)_k,$$

where Φ_α^i is the ith component of mode α, and there is no sum on α. (In the case of algebraic summation the subscript k is not relevant and can be ignored in this equation and in those that follow.)

There are several methods for combining these peak physical responses in the individual modes, $\left(R_\alpha^i\right)_k^{max}$, into estimates of the total peak response, $\left(R^i\right)_k^{max}$. Most of the methods implemented in Abaqus/Standard follow the ASCE 4–98 standard for Seismic Analysis of Safety Related Nuclear Structures and Commentary. The updated documents, "Reevaluation of Regulatory Guidance on Modal Response Combination Methods for Seismic Response Spectrum Analysis" issued in 1999 (NUREG/CR-6645, BNL-NUREG-52276) and "Draft Regulatory Guide" (DG-1127) issued in 2005 contain new recommendations. You are advised to read the new recommendations before choosing a modal summation method from among those described below.

The Absolute Value Method

The absolute value method is the most conservative method for combining the modal responses. It is obtained by summing the absolute values resulting from each mode:

$$\left(R^i\right)_k^{max} = \sum_\alpha \left|\left(R_\alpha^i\right)_k^{max}\right|.$$

This method implies that all of the responses peak simultaneously. It will overpredict the peak response of most systems; therefore, it may be too conservative to help in design.

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=ABS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Absolute values**

The Square Root of the Sum of the Squares Modal Summation Method

The square root of the sum of the squares method is less conservative than the absolute value method. It is also usually more accurate if the natural frequencies of the system are well separated. It uses the square root of the sum of the squares to combine the modal responses:

$$\left(R^i\right)_k^{max} = \sqrt{\sum_\alpha \left(\left(R_\alpha^i\right)_k^{max}\right)^2}.$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=SRSS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Square root of the sum of squares**

The Naval Research Laboratory Method

The absolute value and square root of the sum of the squares methods can be combined to yield the Naval Research Laboratory method. It distinguishes the mode, β, in which the physical variable has its maximum response and adds the square root of the sum of squares of the peak responses in all other modes to the absolute value of the peak response of that mode. This method gives the estimate:

$$\left(R^i\right)_k^{max} = \left|\left(R_\beta^i\right)_k^{max}\right| + \sqrt{\sum_{\alpha \neq \beta} \left(\left(R_\alpha^i\right)_k^{max}\right)^2}.$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=NRL

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Naval Research Laboratory**

The Ten-Percent Method

The ten-percent method recommended by Regulatory Guide 1.92 (1976) is no longer recommended according to the "Reevaluation of Regulatory Guidance on Modal Response Combination Methods for Seismic Response Spectrum Analysis" document issued in 1999. It is retained here because of its extensive prior use. The ten-percent method modifies the square root of the sum of the squares method by adding a contribution from all pairs of modes α and β whose frequencies are within 10% of each other, giving the estimate:

$$\left(R^i\right)_k^{max} = \sqrt{\sum_\alpha \left(\left(R_\alpha^i\right)_k^{max}\right)^2 + 2 \sum_{\alpha<\beta} \left|\left(R_\alpha^i\right)_k^{max} \left(R_\beta^i\right)_k^{max}\right|}.$$

The frequencies of modes α and β are considered to be within 10% of each other whenever

$$\frac{\omega_\beta - \omega_\alpha}{\omega_\beta} \leq 0.1\,, \quad \alpha < \beta.$$

The ten-percent method reduces to the square root of the sum of the squares method if the modes are well separated with no coupling between them.

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=TENP

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Ten percent**

The Complete Quadratic Combination Method

Like the ten-percent method, the complete quadratic combination method improves the estimation for structures with closely spaced eigenvalues. The complete quadratic combination method combines the modal response with the formula

$$\left(R^i\right)_k^{max} = \sqrt{\sum_\alpha \sum_\beta \left(R_\alpha^i\right)_k^{max} \rho_{\alpha\beta} \left(R_\beta^i\right)_k^{max}},$$

where $\rho_{\alpha\beta}$ are cross-correlation coefficients between modes α and β, which depend on the ratio of frequencies and modal damping between the two modes:

$$\rho_{\alpha\beta} = \frac{8\sqrt{\xi_\alpha \xi_\beta}\left(\xi_\alpha + r_{\beta\alpha}\xi_\beta\right) r_{\beta\alpha}^{3/2}}{\left(1-r_{\beta\alpha}^2\right)^2 + 4\xi_\alpha\xi_\beta r_{\beta\alpha}\left(1+r_{\beta\alpha}^2\right) + 4\left(\xi_\alpha^2+\xi_\beta^2\right) r_{\beta\alpha}^2},$$

where $r_{\beta\alpha} = \omega_\beta / \omega_\alpha$.

If the modes are well spaced, their cross-correlation coefficient will be small ($\rho_{\alpha\beta} \ll 1$) and the method will give the same results as the square root of the sum of the squares method.

This method is usually recommended for asymmetrical building systems since, in such cases, other methods can underestimate the response in the direction of motion and overestimate the response in the transverse direction.

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=CQC

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Complete quadratic combination**

The Grouping Method

This method, also known as the NRC grouping method, improves the response estimation for structures with closely spaced eigenvalues. The modal responses are grouped such that the lowest and highest frequency modes in a group are within 10% and no mode is in more than one group. The modal responses are summed absolutely within groups before performing a SRSS combination of the groups. Within the group responses are summed as

$$\left(R_{gr}^i\right)_k^{max} = \left|\left(R_1^i\right)_k^{max}\right| + \left|\left(R_2^i\right)_k^{max}\right| \ldots + \left|\left(R_n^i\right)_k^{max}\right|,$$

for “n” frequencies within any “gr” group and then performing

$$\left(R^i\right)_k^{max} = \sqrt{\sum_{gr=1}^{gr=n}\left(\left(R_{gr}^i\right)_k^{\max}\right)^2}.$$

The above expression includes all the groups; in addition, the group can consist of just one frequency response if this frequency does not have another member that is within the 10% limit.

The ten-percent method will always produce results higher in value than the grouping method.

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=GRP

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Grouping method**

Double Sum Combination

This method, also known as Rosenblueth's double sum combination (*Rosenblueth and Elorduy, 1969*), is the first attempt to evaluate modal correlation based on random vibration theory. It utilizes the time duration t_D of strong earthquake motion. The mode correlation coefficients $c_{\alpha\beta}$, which depend also on the frequencies and damping coefficient ξ, lead to the following mode combination:

$$\left(R^i\right)_k^{max} = \sqrt{\sum_\alpha \sum_\beta \left(R_\alpha^i\right)_k^{max} c_{\alpha\beta}\left(R_\beta^i\right)_k^{max}},$$

where

$$c_{\alpha\beta} = \frac{1}{1+\left(\frac{\omega_{\alpha'}-\omega_{\beta'}}{\xi_{\alpha'}\omega_\alpha+\xi_{\beta'}\omega_\beta}\right)^2},$$

where

$$\omega_{\alpha'} = \omega_\alpha\sqrt{1-\xi_\alpha^2},$$
$$\xi_{\alpha'} = \xi_\alpha + \frac{2}{t_D\omega_\alpha}.$$

Input File Usage: **RESPONSE SPECTRUM*, COMP=*comp*, SUM=DSC

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Double sum combination**

Separation of Modal Responses into Periodic and Rigid Responses

Each spectrum can be divided into low-, medium-, and high-frequency regions. In the low-frequency region the modes are usually uncorrelated. In the mid-range they are partially correlated, and in the high-frequency region they tend to be correlated with the input data and, hence, between themselves. Each of these regions may, therefore, require special consideration. These considerations are addressed by the Gupta and Lindley-Yow methods.

We divide the modal response into two parts: the rigid part and the damped periodic part. It is assumed that the rigid part and the damped periodic part are statistically independent. Let us define the rigid response coefficient for each mode by $0 \le \alpha_\beta \le 1$, the rigid response by $\left(R_\beta^i\right)^r = \alpha_\beta\left(R_\beta^i\right)$, and the periodic response by $\left(R_\beta^i\right)^p = R_\beta^i\sqrt{1-\alpha_\beta^2}$. Therefore, the relationship between both regions becomes:

$$\left(\left(R_\beta^i\right)^p\right)^2 + \left(\left(R_\beta^i\right)^r\right)^2 = 1,$$

which is equivalent to summing both regions according to the SRSS rule. The value of the rigid response coefficient is chosen depending on the method used. For the Gupta method this coefficient is estimated based on two user-specified frequencies: F_1 defining the beginning of the periodic region, and F_2 defining the beginning of the rigid region (which is often equivalent to the frequency associated with zero period acceleration). For the current frequency F_β, this expression for the Gupta method is

$$\alpha_\beta = \frac{\log\left(\frac{F_\beta}{F_1}\right)}{\log\left(\frac{F_2}{F_1}\right)}.$$

For the Lindley-Yow method this coefficient is equal to

$$\alpha_\beta = \frac{ZPA}{S_{\alpha\beta}},$$

where ZPA denotes the zero period acceleration for the user-specified frequency value and $S_{\alpha\beta}$ denotes the acceleration spectrum for mode β. The Lindley-Yow method does not work well for the region of frequencies on the low-frequency end of the spectrum. Therefore, we set the coefficient $\alpha_\beta = 0$ for all modes below the frequency of the peak acceleration value for this spectrum.

The rigid response method works with only two modal summation rules: CQC and DSC with their respective definitions for the mode correlation coefficients $c_{\alpha\beta}$. By default, no rigid response is included. To include a rigid response, you must specify the rigid response method.

Input File Usage: Use the following option to specify the Gupta method for the rigid response:

**RESPONSE SPECTRUM*, COMP=*comp*, RIGID RESPONSE=GUPTA

Use the following option to specify the Lindley-Yow method for the rigid response:

**RESPONSE SPECTRUM*, COMP=*comp*, RIGID RESPONSE=LINDLEY-YOW

Abaqus/CAE Usage: You cannot specify a rigid response in Abaqus/CAE.

Selecting the Modes and Specifying Damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the Modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS
**SELECT EIGENMODES*, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying Damping

Damping is almost always specified for a mode-based procedure; see *Material Damping*. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for an mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

**MODAL DAMPING*, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

**MODAL DAMPING*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Damping**: **Specify damping over ranges of**: **Modes**

Use the following input to define damping by specifying a frequency range:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Damping**: **Specify damping over ranges of**: **Frequencies**

Example of Specifying Damping

Figure 1 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

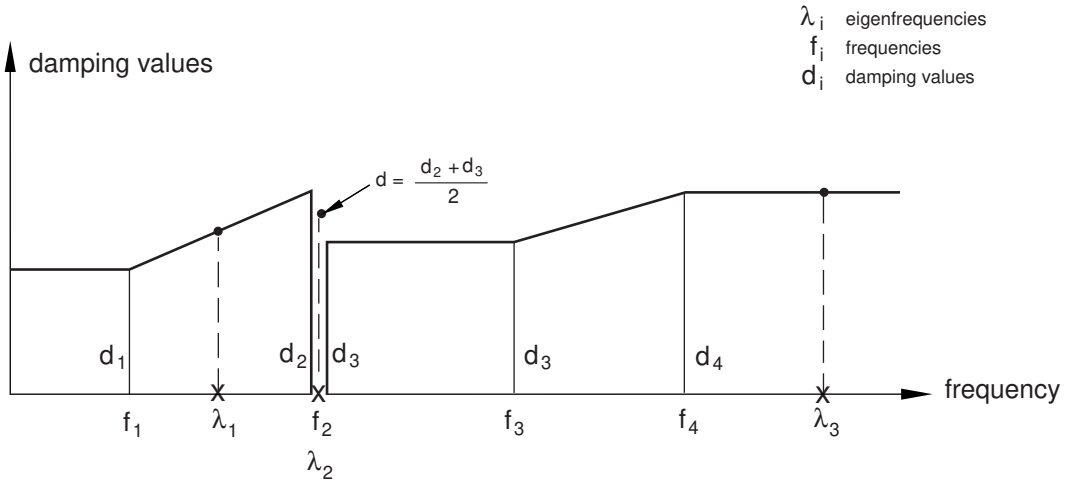


Figure 1: Damping values specified by frequency range.

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
f1, d1
f2, d2
f2, d3
f3, d3
f4, d4
```

Rules for Selecting Modes and Specifying Damping Coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.

- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Using the Missing Mass Method

You can use the missing mass method to augment the missing inertia in the dynamic response when the number of eigenmodes included in the superposition does not sufficiently represent the total mass of the structure. The missing inertia is due primarily to the truncated higher frequency modes that belong to the in-phase (rigid) response. Quasistatic residual modes in the direction of applied spectra can be substituted for these truncated modes. This technique is most suitable for use with the response spectra summations that split the frequency range into periodic and rigid phases, such as the Gupta or Lindley-Yow methods. However, any modal summation method can be used with the missing mass method. The missing mass method results in better response accuracy and is explained in three small spring-mass examples introduced first in the paper by Dhileep (*Dhileep and Bose, 2008*). The residual vector representing missing energy is found by a simple inertia balance in eigenspace:

$$K x_0 = -M\left(u_B - \sum_{i=1}^{m} \Phi_i \Gamma_i\right) a_g,$$

where

K
: represents the stiffness of the model,

M
: represents the mass of the model,

Φ_i and Γ_i
: represent the i^{th} out of m eigenvectors and participation factors,

u_B
: is the unit base displacement in the direction of the earthquake spectrum,

a_g
: is a ground acceleration, and

x_0
: is a pseudostatic residual mode representing the response of the remaining modes beyond the mode m.

Input File Usage: **RESPONSE SPECTRUM*, MISSING MASS METHOD

Abaqus/CAE Usage: The missing mass method is not supported in Abaqus/CAE.

Initial Conditions

It is not appropriate to specify initial conditions in a response spectrum analysis.

Boundary Conditions

All points constrained by boundary conditions and the ground nodes of connector elements are assumed to move in phase in any one direction. This base motion can use a different input spectrum in each of three orthogonal directions (two directions in a two-dimensional model). You define the input spectra, $S(\omega, \xi)$, as functions of frequency, ω, for different values of critical damping, ξ, as described earlier in *Specifying a Spectrum*. Secondary bases cannot be used in a response spectrum analysis.

Loads

The only "loading" that can be defined in a response spectrum analysis is that defined by the input spectra, as described earlier. No other loads can be prescribed in a response spectrum analysis.

Predefined Fields

Predefined fields, including temperature, cannot be used in response spectrum analysis.

Material Options

The density of the material must be defined (*Density*). The following material properties are not active during a response spectrum analysis: plasticity and other inelastic effects, rate-dependent material properties, thermal properties, mass diffusion properties, electrical properties, and pore fluid flow properties—see *General and Perturbation Procedures*.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard can be used in a response spectrum analysis—see *Choosing the Appropriate Element for an Analysis Type*.

Output

All the output variables in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*. The value of an output variable such as strain, E; stress, S; or displacement, U, is its peak magnitude.

In addition to the usual output variables available, the following modal variables are available for response spectrum analysis and can be output to the data and/or results files (see *Output to the Data and Results Files*):

GU Generalized displacements for all modes.

GV Generalized velocities for all modes.

GA Generalized accelerations for all modes.

SNE Elastic strain energy for the entire model per each mode.

KE Kinetic energy for the entire model per each mode.

T External work for the entire model per each mode.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in response spectrum analysis. However, whole model variables such as ALLIE (total strain energy) are available for modal-based procedures as output to the data and/or results files (see *Output to the Data and Results Files*).

Reaction force output is not supported for response spectrum analysis using eigenmodes extracted using the default SIM-based frequency extraction procedure with either the AMS or Lanczos eigensolver. Reaction force output in response spectrum analysis using eigenmodes extracted with the non-SIM-based Lanczos eigensolver provides directional combinations of so-called, modal reaction forces weighted with maximal absolute values of corresponding generalized displacements. Directional and modal combination rules used for the reaction force calculation are the same as for other nodal output variables. Modal reaction forces are calculated in the frequency extraction procedure. They represent static reaction forces calculated for the normal mode shapes. Generally, they cannot adequately represent reaction force in dynamic analysis. For models with diagonal mass and diagonal damping matrices the superposition of the modal reaction forces can provide a reasonable approximation of a nodal reaction force in mode-based analyses other than response spectrum analysis. In response spectrum analysis the model response can be better represented by requesting section stresses and section forces in structural elements containing supported nodes.

Input File Template

```
*HEADING
…
*BOUNDARY
Data lines to define points to be excited by the base motion controlled by the input spectra
*SPECTRUM, NAME=name1, TYPE=type
Data lines to define spectrum “name1” as a function of frequency, ω, and
fraction of critical damping, ξ
*SPECTRUM, NAME=name2, TYPE=type
Data lines to define spectrum “name2” as a function of frequency, ω, and
fraction of critical damping, ξ
**
*STEP
*FREQUENCY
Data line to specify number of modes to be extracted
*END STEP
**
*STEP
*RESPONSE SPECTRUM, COMP=comp, SUM=sum
Data lines referring to response spectra and defining direction cosines
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data lines to define modal damping
*END STEP
```

References

- Dhileep, M., and P. R. Bose, “A Comparative Study of "Missing Mass" Correction Methods for Response Spectrum Method of Seismic Analysis,” *Computers and Structures*, vol. 86, pp. 2087–2094, 2008.
- Rosenblueth, E., and J. Elorduy, “Response of Linear Systems to Certain Transient Disturbances,” *Proceedings of the Fourth World Conference on Earthquake Engineering, Santiago, Chile*, 1969.

Random Response Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A random response analysis:

- is a linear perturbation procedure that gives the linearized dynamic response of a model to user-defined random excitation; and
- uses the set of modes extracted in a previous eigenfrequency extraction step to calculate the power spectral densities of response variables (stresses, strains, displacements, etc.) and the corresponding root mean square (RMS) values of these same variables.

References:

- *Defining an Analysis*
- *General and Perturbation Procedures*
- *About Dynamic Analysis Procedures*
- **RANDOM RESPONSE*
- **PSD-DEFINITION*
- **CORRELATION*
- *Configuring a random response procedure*

Random Response Analysis

Random response analysis predicts the response of a system that is subjected to a nondeterministic continuous excitation that is expressed in a statistical sense by a cross-spectral density matrix. Since the loading is nondeterministic, it can be characterized only in a statistical sense; Abaqus/Standard assumes that the excitation is stationary and ergodic. These statistical measures are explained in detail in *Random response analysis*. The random response procedure can, for example, be used to determine the response of an airplane to turbulence, the response of a car to road surface imperfections, the response of a structure to jet noise, or the response of a building to an earthquake.

In the most general case the excitation is defined as a frequency-dependent cross-spectral density (CSD) matrix. Except in cases involving moving noise or user subroutine *UCORR*, it is assumed that for a given load case the CSD matrix can be separated into a product of a frequency-dependent, complex-valued scalar function and a frequency-independent, complex-valued, spatial correlation matrix. This assumption helps reduce both the computational time and the amount of required user input but implies that each element of the CSD matrix in a given load case has the same frequency dependence. You can define a different frequency dependence for each load case, but the loads in one load case will not be correlated with loads in another. Consequently, the system CSD matrix is assembled by simply summing (superimposing) the CSD matrices of the individual load cases.

The frequency-dependent scalar function can be composed of a weighted sum of user-defined, complex-valued, frequency functions. These user-defined frequency functions are specified in units of power spectral density. You assign weights to each frequency function as well as specify properties of the spatial correlation matrix that defines the correlation between excitations at different locations and in different directions for a particular load case. Frequency functions and correlations are discussed below; see *Defining the Frequency Functions* and *Defining the Correlation*.

The loads can be defined as concentrated point loads, as distributed loads, as connector element loads, or as base motion excitations, as described below in *Boundary Conditions* and *Loads*. Multiple, uncorrelated load cases can be defined for concentrated point loads, connector loads, and base motions. Load case 1 is reserved for all distributed loads defined in a particular step. In these steps load case 1 cannot be used for any concentrated point load, connector load, or base motion. Thus, there cannot be any correlation between distributed loads and any other load. Moreover, base motion excitations are assumed to be statistically independent (no correlation) with any other load type even when the same load case number is used. The concentrated point and connector element loads are assumed to be correlated if the same load case number is used.

The random response procedure is based on using a subset of the modes of the system, which must first be extracted by using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part. The model can be preloaded prior to the eigenfrequency extraction. Initial stress effects are included in the stiffness used in the eigenfrequency extraction if geometric nonlinearities are included in the general analysis procedure used to apply the preloads (*General and Perturbation Procedures*).

The random response of the model is expressed as power spectral density values of nodal and element variables, as well as their root mean square values.

Defining the Frequency Range

You specify the frequency range of interest for the random response procedure. The response is calculated at multiple points between the lowest frequency of interest and the first eigenfrequency in the range, between each eigenfrequency in the range, and between the last eigenfrequency in the range and the highest frequency in the range as illustrated in *Figure 1*. The default number of calculation points in each interval is 20; you can change this number when you define the step. Accurate RMS values can be obtained only if enough points are used so that Abaqus/Standard can integrate accurately over the frequency range. The bias function allows the points on the frequency scale to be spaced closer together at the eigenfrequencies, thus allowing detailed definition of the response close to resonant frequencies and more accurate integration.

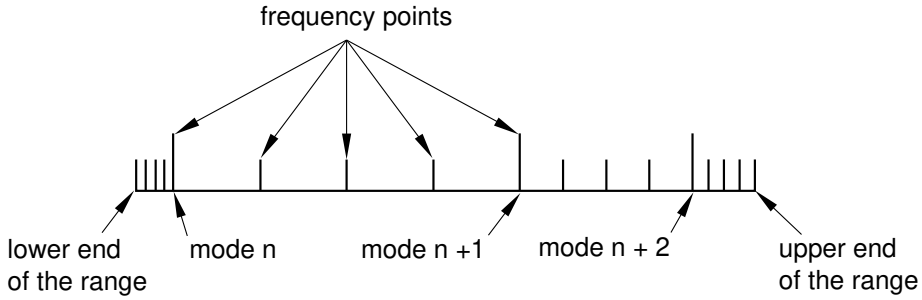


Figure 1: Division of range using modes and 5 calculation points.

Input File Usage: **RANDOM RESPONSE*
lower_freq_limit, upper_freq_limit, num_calc_pts, bias_parameter, freq_scale

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Random response**

The Bias Parameter

The bias parameter can be used to provide closer spacing of the result points either toward the middle or toward the ends of each frequency interval. *Figure 2* shows a few examples of the effect of the bias parameter on the frequency spacing.

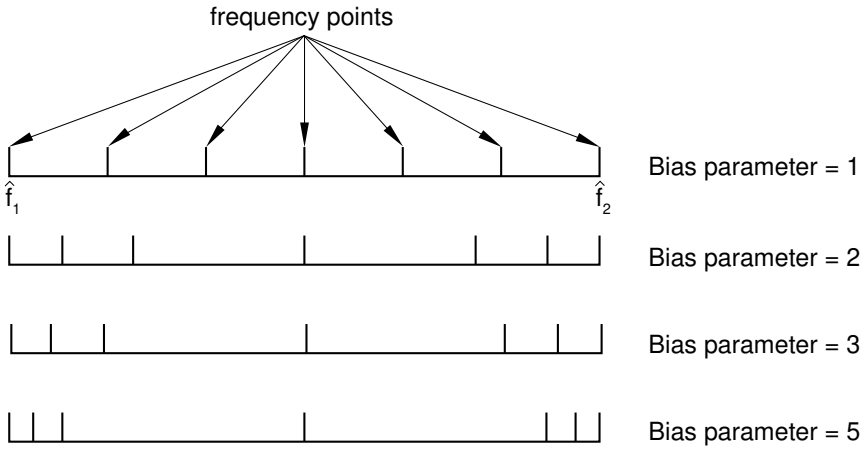


Figure 2: Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used to calculate the frequency at which results are presented is as follows:

$$\hat{f}_k = \tfrac{1}{2}\left(\hat{f}_1 + \hat{f}_2\right) + \tfrac{1}{2}\left(\hat{f}_2 - \hat{f}_1\right)|y|^{1/p}\,\mathrm{sign}\,(y)\,,$$

where

y

$= -1 + 2\,(k-1)\,/\,(n-1)$;

n

is the number of frequency points at which results are to be given;

k

is one such frequency point ($k = 1, 2, \ldots, n$);

$\hat{f}_1$

is the lower limit of the frequency interval;

$\hat{f}_2$

is the upper limit of the interval;

$\hat{f}_k$

is the frequency at which the *k*th results are given;

p

is the bias parameter value; and

$\hat{f}$

is the frequency or the logarithm of the frequency, depending on the chosen frequency scale.

A bias parameter, *p*, that is greater than 1.0 provides closer spacing of the results points toward the ends of each frequency interval (as shown in the examples above), while values of *p* that are less than 1.0 provide closer spacing toward the middle of each frequency interval. The default value of the bias parameter for random response analysis is 3.0.

Defining the Frequency Functions

To define the random loading, you specify a frequency function and a cross-correlation definition that refers to the frequency function. The frequency functions are defined as model data (that is, they are step independent) and must be named. A log-log scale is used in interpolating between the given values.

The type of units in the CSD matrix of the excitation are specified as part of the frequency function definition. The default type is power units. If the CSD matrix of the excitation is due to base motion, the units must be in *g* units and you should define the acceleration of gravity. Alternatively, decibel units can be specified; this type of units is explained below.

Input File Usage: Use one of the following options to define the frequency function:

**PSD-DEFINITION*, NAME=*name*, TYPE=FORCE (default; power units)
**PSD-DEFINITION*, NAME=*name*, TYPE=BASE, G=*g*
**PSD-DEFINITION*, NAME=*name*, TYPE=DB, DB REFERENCE=P_{ref}

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power**, **Decibel**, or **Gravity**

Defining the Cross-Spectral Density Matrix in Decibel Units

In Abaqus/Standard the decibel value $db\,(f)$ is related to the frequency function $P\,(f)$ by the following full octave band conversion formula:

$$db\,(f) = 10 \log_{10} \frac{P(f)}{\sqrt{2} P_{\text{ref}} / f_c},$$

where P_{ref} is the user-specified reference power and f_c is the midband frequency (see *Table 1*).

Table 1: Standard octave bands.

Band number	Band center (frequency, Hz)
1	1.0
2	2.0
3	4.0
4	8.0
5	16.0
6	31.5
7	63.0

Band number	Band center (frequency, Hz)
8	125.0
9	250.0
10	500.0
11	1000.0
12	2000.0
13	4000.0
14	8000.0
15	16000.0

Hence, the frequency function follows from the function defined in decibel units as

$$P(f) = \frac{\sqrt{2}}{f_c} P_{\text{ref}} 10^{db(f)/10}.$$

If you have data in terms of an alternative frequency scale (for example, one-third octave band), an equivalent full octave band power reference value can be obtained as described in *Random response analysis*.

$P(f)$ in decibels must be specified as a function of the frequency band; the associated midband frequencies are given in *Table 1*.

Alternate Methods for Defining Frequency Functions

You can define a frequency function in an external file or in a user subroutine.

Defining the Frequency Function in an External File

The data to define a frequency function can be contained in an external file.

Input File Usage: **PSD-DEFINITION*, NAME=*name*, TYPE=*type*, INPUT=*file name*

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power**, **Decibel**, or **Gravity**; **Real**, **Imaginary**, **Frequency**

Defining the Frequency Function in a User Subroutine

Complicated frequency functions can be more easily defined by user subroutine *UPSD* than by entering data directly.

Input File Usage: **PSD-DEFINITION*, NAME=*name*, TYPE=*type*, USER

Any data lines given will be ignored if the USER parameter is specified.

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power** or **Gravity**; toggle on **Specify data in an external user subroutine**

Defining the Correlation

You define the cross-correlation between the applied nodal loads or base motions. You can also assign scaling (weight) factors to the frequency functions through the cross-correlation definition. Distributed loads are converted

to equivalent nodal loads, which are treated as individual point loads with respect to the cross-correlation. The cross-correlation is defined in the random response step and references a particular load case number and frequency function.

Three types of correlation can be defined: correlated, uncorrelated, and moving noise. As many correlations as needed to define the random loading can be specified unless the moving noise type is chosen, in which case only one correlation can appear in the step definition.

- For the correlated type all terms in the cross-spectral density matrix are considered, which implies that the loads on all degrees of freedom within the load case are fully correlated (statistically dependent on each other).
- For the uncorrelated type only diagonal terms in the cross-spectral density matrix are considered, which implies that no correlation exists between the load on one degree of freedom and the load on another. You should exercise caution when choosing the uncorrelated type with distributed loads since the equivalent nodal forces would be uncorrelated with each other (statistically independent).
- For the moving noise type, the terms in the correlation matrix depend on the relative position of the points where the loads are applied. This type can be used only in conjunction with concentrated point loads and distributed loads. In addition, the moving noise formulation assumes that the frequency function referenced by the cross-correlation defines a reference power spectral density function of the noise source. (It is a reference power spectral density because it can later be scaled by the magnitude of the loadings specified as distributed, concentrated point, or connector element loads.) Since the power spectral density is real-valued for real-valued variables, the frequency function must not contain imaginary terms when used with the moving noise type of cross-correlation.

Input File Usage: Use one of the following options to define the correlation:

**CORRELATION*, TYPE=CORRELATED, PSD=*name*
**CORRELATION*, TYPE=UNCORRELATED, PSD=*name*
**CORRELATION*, TYPE=MOVING NOISE

For the moving noise type the reference to the power spectral density function must be given on each data line.

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **Correlated** or **Uncorrelated**; **PSD**: *psd_amplitude_name*

Specifying Whether the Correlation Matrix Is Complex

For correlated or uncorrelated cross-correlations you can specify whether or not both real and imaginary terms will be included in the spatial correlation matrix. This specification does not affect the imaginary terms given for the power spectral density frequency function.

Input File Usage: Use one of the following options:

**CORRELATION*, TYPE=CORRELATED, COMPLEX=YES or NO, PSD=*name*
**CORRELATION*, TYPE=UNCORRELATED, COMPLEX=YES or NO, PSD=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **Correlated** or **Uncorrelated**; **PSD**: *psd_amplitude_name*; **Real**; **Imaginary**

Alternate Methods for Defining a Correlation

You can define a correlation in an external input file or in a user subroutine.

Defining the Correlation in an External Input File

The data to define a correlation can be contained in an external input file.

Input File Usage: **CORRELATION*, TYPE=*type*, PSD=*name*, INPUT=*file_name*

Abaqus/CAE Usage: You cannot define a correlation in an external file in Abaqus/CAE.

Defining the Correlation in a User Subroutine

Simple excitations, such as uncorrelated white noise, are easily defined. Excitations involving more complicated correlations, including cases where the elements of the CSD matrix have different frequency dependencies, can be defined through user subroutine *UCORR*. If the user subroutine is specified, only the load case number must be entered as part of the correlation definition. A user subroutine cannot be used to define a moving noise correlation.

For uncorrelated cross-correlations only the diagonal terms of the correlation matrix specified in *UCORR* will be used. The combination of the cross-correlation with the various kinds of applied loads is discussed in more detail below.

Input File Usage: Use one of the following options:

**CORRELATION*, TYPE=CORRELATED, USER, COMPLEX=YES or NO, PSD=*name*
**CORRELATION*, TYPE=UNCORRELATED, USER, PSD=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **User**

Selecting the Modes and Specifying Damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the Modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

**SELECT EIGENMODES*, DEFINITION=MODE NUMBERS
**SELECT EIGENMODES*, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying Damping

Damping is almost always specified for a random response analysis (see *Material Damping*). If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in *Material Damping*. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

**MODAL DAMPING*, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

**MODAL DAMPING*, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping:

Step module: **Create Step**: **Linear perturbation**: **Random response**: **Damping**

Example of Specifying Damping

Figure 3 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
f1, d1
f2, d2
f2, d3
f3, d3
f4, d4
```

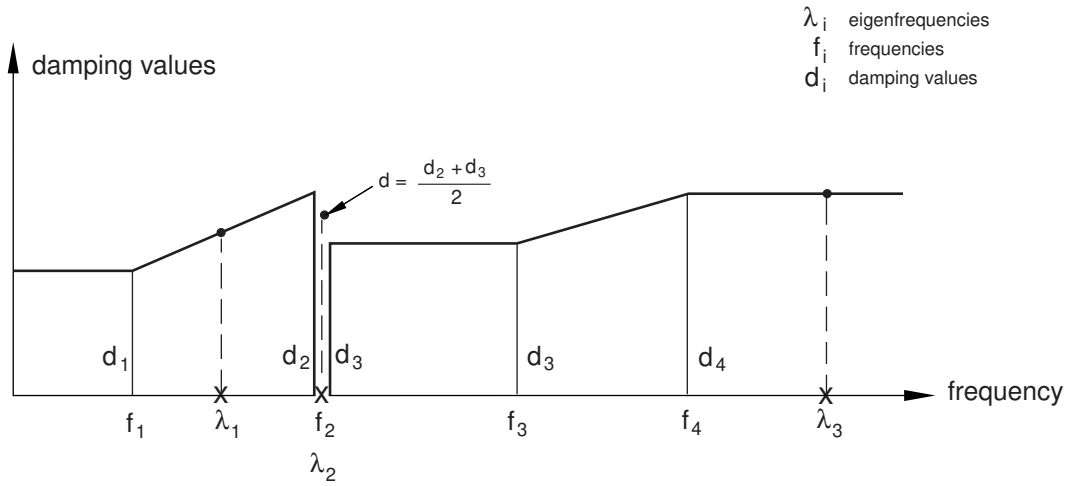


Figure 3: Damping values specified by frequency range.

Rules for Selecting Modes and Specifying Damping Coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Initial Conditions

It is not appropriate to specify initial conditions in a random response analysis.

Boundary Conditions

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions (*Boundary Conditions*) in mode-based dynamic response procedures. Therefore, in a random response analysis the motion of nodes can be specified only as base motion; nonzero displacement, velocity, or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. In addition, any amplitude definitions are ignored in a random response analysis.

The method for prescribing motion in modal superposition procedures is described in *Transient Modal Dynamic Analysis*. In random response analysis only a single (primary) base can be defined.

Defining Multiple Load Cases

The excitation defined by the base motion is assigned to numbered load cases. These load cases are then referenced in the cross-correlation definition. The load cases are associated with frequency functions through the reference in the cross-correlation definition. Any number of load cases can be defined, but load case number 1 cannot be used if distributed loads are defined in the same step.

Input File Usage: **BASE MOTION*, LOAD CASE=*n*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category:** **Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**

Converting Base Motion Excitation to a Cross-Spectral Density Matrix

When the excitation is provided by a base motion, it is converted directly into a cross-spectral density matrix projected onto the eigenspace through the modal participation factors (see *Natural Frequency Extraction*), giving

$$S_{\alpha\beta}(f) = \begin{cases} \sum_J g_J^2 P^J(f) \sum_i \sum_j \Gamma_\alpha^i \Gamma_\beta^j \sum_I \Psi_{ij}^{IJ} (2\pi f)^{2\lambda_I} & \text{for } \alpha < \beta, \\ [S_{\beta\alpha}(f)]^* & \text{for } \alpha > \beta, \\ \mathrm{Re}\left(\sum_J g_J^2 P^J(f) \sum_i \sum_j \Gamma_\alpha^i \Gamma_\beta^j \sum_I \Psi_{ij}^{IJ} (2\pi f)^{2\lambda_I} \right) & \text{for } \alpha = \beta, \end{cases}$$

where the superscript * denotes complex conjugate and where

Γ_α^i

is the modal participation factor for mode α in excitation direction i (i=1–6);

$P^J(f)$

is the frequency function referenced by the Jth cross-correlation and defined as a function of the frequency f in g units;

Ψ_{ij}^{IJ}

is a matrix of weight factors indicating the fraction of P^J to be associated with the correlation between base motion in directions i and j for load case I, as described below;

λ_I

$= 0$, 1, or 2, depending on whether the base motion corresponding to load case I is defined in terms of an acceleration spectrum, a velocity spectrum, or a displacement spectrum (see *Transient Modal Dynamic Analysis*); and

g_J

is the user-specified acceleration of gravity for the same power spectral density frequency function that defines P^J.

If the cross-correlation is defined in user subroutine *UCORR*, Ψ_{ij}^{IJ} is defined in the user subroutine. Otherwise,

Ψ_{ij}^{IJ}

$= a^{IJ}$ for all (i, j) if the excitation is correlated or

$= a^{IJ}\delta_{ij}$ if the excitation is uncorrelated,

where a^{IJ} is the (complex) value of the weight factor by which to scale the frequency function P^J used in load case *I*.

Loads

The loading for random response analysis is defined in general terms by the cross-spectral density matrix $S_{(N,i)(M,j)}(f)$, where *f* is frequency in cycles per time and the subscripts (N, i) and (M, j) refer to degree of freedom *i* at node *N* and degree of freedom *j* at node *M*, respectively. Distributed loads are converted to equivalent nodal loads, which—for the formulation of the correlation matrix—are treated in the same way as concentrated point loads. The units of $S_{(N,i)(M,j)}(f)$ depend on degrees of freedom *i* and *j*.

- If both *i* and *j* are translations, the units are (force)^2(frequency)$^{-1}$; square of force per frequency.
- If both *i* and *j* are rotations, the units are (moment)^2(frequency)$^{-1}$; square of moment per frequency.
- If *i* and *j* represent one translation and one rotation, the units are (force)(moment)(frequency)$^{-1}$.

In addition, any amplitude references on the concentrated point, connector element, or distributed load definitions are ignored in a random response analysis.

Loads in a random response analysis have three attributes: a force defined by a concentrated or distributed load (or a base motion), a power spectral density defined by a frequency function, and correlation coefficients between degrees of freedom. The correlation coefficients are nondimensional, but the force and power spectral density values can have units assigned to them. To maintain the correct units for the random loading, you must define the units in one of the following ways:

- Define nondimensional loads, and define the frequency function using power units: (force)^2(frequency)$^{-1}$ or (moment)^2(frequency)$^{-1}$.
- Define loads using force and moment units, and define the frequency function curve using (frequency)$^{-1}$ units. This is the only option to define loads including forces and moments simultaneously.
- Define base motion using the frequency function curve as (acceleration)^2 (frequency)$^{-1}$, and define the reference gravity acceleration as nondimensional.
- Define base motion using the frequency function curve as (frequency)$^{-1}$, and define the reference gravity acceleration using acceleration units.

Defining Multiple Load Cases

Distributed loads will be assigned automatically to load case number 1. You assign a concentrated point load or connector element load to a numbered load case. Any number of concentrated point and connector element load cases can be specified, but load case number 1 cannot be used for a concentrated point or connector element

load if a distributed load is present in the same step. The concentrated point, connector element, and distributed load cases are associated with frequency functions through the cross-correlation definition.

Input File Usage: Use one or more of the following options:

**CLOAD*, LOAD CASE=*n*
**CONNECTOR LOAD*, LOAD CASE=*m*
**DLOAD*
**DSLOAD*

Correlated and Uncorrelated Loading

For correlated or uncorrelated cross-correlations, the cross-spectral density matrix is defined as

$$S_{(N,i)(M,j)}(f) = \begin{cases} \sum_J P^J(f) \sum_I C^{IJ}_{(N,i)(M,j)} F^I_{(N,i)} F^I_{(M,j)} & \text{for } (N,i) < (M,j), \\ \left[S_{(M,j)(N,i)}(f)\right]^* & \text{for } (N,i) > (M,j), \\ \mathrm{Re}\left(\sum_J P^J(f) \sum_I C^{IJ}_{(N,i)(M,j)} F^I_{(N,i)} F^I_{(M,j)}\right) & \text{for } (N,i) = (M,j), \end{cases}$$

where the superscript * denotes complex conjugate and where

$F^I_{(N,i)}$

is the load magnitude applied to degree of freedom *i* at node *N* for load case *I*;

$P^J(f)$

is the frequency function referenced by the *J*th cross-correlation and defined as a function of the frequency *f* in power (force) or decibel units; and

$C^{IJ}_{(N,i)(M,j)}$

is a matrix of weight factors indicating the fraction of P^J to be associated with the (N,i) (M,j) cross-correlation term for load case *I*, as described below.

If the cross-correlation is defined in user subroutine *UCORR*, $C^{IJ}_{(N,i)(M,j)}$ is defined in the user subroutine. Otherwise,

$C^{IJ}_{(N,i)(M,j)}$

$= a^{IJ}$ for all (N,M,i,j) if the excitation is correlated or

$= a^{IJ}\delta_{NM}\delta_{ij}$ if the excitation is uncorrelated,

where a^{IJ} is the (complex) value of the weight factor by which to scale the frequency function P^J used in load case *I*.

Moving Noise Loading

For moving noise cross-correlations, the cross-spectral density matrix is defined as

$$S_{(N,i)(M,j)}(f) = \sum_I P^I(f) \exp\left(i2\pi f(\mathbf{x}_N - \mathbf{x}_M) \cdot \mathbf{v}^I \frac{1}{\mathbf{v}^I \cdot \mathbf{v}^I}\right) F^I_{(N,i)} F^I_{(M,j)},$$

where

$F^I_{(N,i)}$

is the load magnitude applied to degree of freedom i at node N for load case I;

$P^I(f)$

is the reference power spectral density function associated with load case I and defined as a function of the frequency f in power (force) or decibel units;

$\mathbf{v}^I$

is the velocity vector of noise propagation given for load case I; and

$\mathbf{x}_N$

are the coordinates of node N.

This definition of moving noise implies that the different noise sources have no cross-correlation. Therefore, it is most generally used with only one noise source ($I = 1$ only). In addition, since $P^I(f)$ is the actual power spectral density of the moving noise source, it must be defined as a real-valued function.

Predefined Fields

Predefined fields, including temperature, cannot be used in random response analysis.

Material Options

As in any dynamic analysis procedure, mass or density (*Density*) must be assigned to some regions of any separate parts of the model where dynamic response is required. The following material properties are not active during a random response analysis: plasticity and other inelastic effects, rate-dependent properties, thermal properties, mass diffusion properties, electrical properties, and pore fluid flow properties (see *General and Perturbation Procedures*).

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard can be used in a random response analysis (see *Choosing the Appropriate Element for an Analysis Type*).

Output

In random response analysis the value of a variable is its power spectral density; all of the output variables in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*. Power spectral density values are not available for concentrated and distributed loads and for SINV.

Options are also provided in random response analysis to obtain root mean square values for certain variables, as listed below. Total values include base motion, while relative values are measured relative to the base motion.

Element integration point variables:

RS Root mean square of all stress components.

RE Root mean square of all strain components.

Element nodal point variables:

MISES Mises equivalent stress..

RMISES Root mean square of Mises equivalent stress.

For connector elements, the following element output variables are available:

RCTF Root mean square of connector total forces.

RCEF Root mean square of connector elastic forces.

RCVF Root mean square of connector viscous forces.

RCRF Root mean square of connector reaction forces.

RCSF Root mean square of connector friction forces.

RCU Root mean square of connector relative displacements.

RCCU Root mean square of connector constitutive displacements.

Nodal variables:

RU Root mean square values of all components of the relative displacement/rotation at a node.

RTU Root mean square values of all components of the total displacement/rotation at a node.

RV Root mean square values of all components of the relative velocity at a node.

RTV Root mean square values of all components of the total velocity at a node.

RA Root mean square values of all components of the relative acceleration at a node.

RTA Root mean square values of all components of the total acceleration at a node.

RRF Root mean square values of all components of reaction forces and reaction moments at a node.

No energy values are available for a random response analysis.

To reduce the computational cost of random response analysis, you should request output only for selected element and node sets. Abaqus/Standard will calculate the response for only the element and nodal variables requested.

In a SIM-based random response procedure, when you request MISES or RMISES output, Abaqus/Standard computes the responses and stores them in the output database (`.odb`) file. Abaqus/Viewer does not compute these responses; it reads and visualizes them from the output database (`.odb`) file. Element stress output in the frequency step preceding the random response step is not required in a SIM-based random response procedure. If MISES or RMISES output for a selected set of elements is required, you must specify the name of that element set for the element stress output request in the random response procedure. Abaqus/Standard supports computation of MISES and RMISES variables at the integration points, element centroids, and element nodes. You can also request MISES and RMISES variables to be averaged at the nodes.

By contrast, in a non-SIM–based random response procedure, when you request MISES or RMISES output, Abaqus/Standard stores the necessary data in the output database (`.odb`) file and Abaqus/Viewer does the actual computation of the responses. These computations require element stress output in the frequency step preceding the random response step. Specifying the name of the element set in the output request in the random response step has no effect on these two output variables. If MISES or RMISES output for a selected set of elements is required, you must specify the name of that element set for the element stress output request in the preceding frequency step. Unlike in other procedures, MISES and RMISES output for random response analysis is computed at the element nodal points and not at the element integration points.

Input File Template

```
*HEADING
…
*PSD-DEFINITION, NAME=name, TYPE=type
Data lines to define a frequency function (or PSD function for moving noise)
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*END STEP
*STEP
*RANDOM RESPONSE
Data line to specify frequency range of interest
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data line to define modal damping
*CORRELATION, PSD=name, TYPE=type
Data lines to specify correlation for various excitation load cases (n, p)
*DLOAD
Data lines to define distributed loads
*CLOAD, LOAD CASE=n
Data lines to define concentrated loads in load case n
*CONNECTOR LOAD, LOAD CASE=m
```

Data lines to define connector loads in load case m
**BASE MOTION*, DOF=*dof*, LOAD CASE=*p*
Data lines to define base motion p
**END STEP*

Steady-State Transport Analysis

Products: Abaqus/Standard

Overview

A steady-state transport analysis:

- allows for steady-state rolling and sliding solutions including frictional effects and inertia effects;
- allows for steady-state solutions to be obtained directly or by using a quasi-steady-state (pass-by-pass) technique;
- is used to model the interaction between a deformable rolling object and one or more flat, convex, or concave surfaces;
- is based on a specialized analysis capability where the rigid body motion is described in a spatial or Eulerian manner and the deformation in a material or Lagrangian manner;
- allows for one element set in a model to be described in an Eulerian manner while the rest of the elements in the model are treated in a classical Lagrangian manner;
- can be preceded by a static stress analysis or followed by a natural frequency extraction or a complex eigenvalue extraction step;
- uses regular stress/displacement elements and special steady-state rolling and sliding contact pairs;
- is currently available only for three-dimensional analysis with an axisymmetric geometry or a periodic geometry; and
- allows rate-independent, rate-dependent, or history-dependent material behavior.

References:

- *Defining an Analysis*
- *Symmetric Model Generation*
- *Using Abaqus/Standard Output Variable Identifiers*
- **STEADY STATE TRANSPORT*
- **SYMMETRIC MODEL GENERATION*
- **MOTION*
- **TRANSPORT VELOCITY*
- **ACOUSTIC FLOW VELOCITY*

Steady-State Transport Analysis

It is cumbersome to model rolling and sliding contact, such as a tire rolling along a rigid surface or a disc rotating relative to a brake assembly, using a traditional Lagrangian formulation since the frame of reference in which motion is described is attached to the material. An observer in this reference frame views even steady-state rolling as a time-dependent process since each point undergoes a repeated history of deformation. Such an analysis is computationally expensive since a transient analysis must be performed and fine meshing is required along the entire surface of the cylinder.

The steady-state transport analysis capability in Abaqus/Standard uses a reference frame that is attached to the axle of the rotating cylinder. An observer in this frame sees the cylinder as points that are not moving, although

the material of which the cylinder is made is moving through those points. This removes the explicit time dependence from the problem—the observer sees a fixed point anywhere, with material moving through it. Thus, the finite element mesh describing the cylinder in this frame of reference does not undergo the large rigid body spinning motion. This means that a fine mesh is required only near the contact zone.

This description can be viewed as a mixed Lagrangian/Eulerian method, where rigid body rotation is described in a spatial or Eulerian manner, and deformation, which is now measured relative to the rotating rigid body, is described in a material or Lagrangian manner. It is this kinematic description that converts the steady-state moving contact problem into a purely spatially dependent simulation.

The steady-state rolling and sliding analysis capability provides solutions that include frictional effects, inertia effects, and material convection for most rate-independent, rate-dependent, and history-dependent material models.

The theory is described in detail in *Steady-state transport analysis*.

Input File Usage: **STEADY STATE TRANSPORT*

Pass-by-Pass Analysis Technique

By default, the steady-state transport analysis procedure in Abaqus/Standard solves for a steady-state rolling and sliding solution directly as a series of increments, with iterations to obtain equilibrium within each increment. The solution in each increment is a steady-state solution corresponding to the loads acting on the structure at that instant. The steady-state transport analysis procedure also provides an alternative technique to obtain a quasi-steady-state rolling and sliding solution as a series of increments, with iterations to obtain equilibrium within each increment. However, the solution in each increment is usually not a steady-state solution corresponding to the loads acting on the structure at that instant. A steady-state solution is generally obtained in several increments, with each increment corresponding to a loading pass through the structure. Each loading pass through the structure can have a different magnitude.

The pass-by-pass analysis technique is relevant only when used with plasticity/creep models. It has no effect on a viscoelastic material model.

Input File Usage: **STEADY STATE TRANSPORT*, PASS BY PASS

Unstable Problems

Local instabilities (e.g., surface wrinkling, material instability, or local buckling), can occur in a steady-state transport analysis. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Defining the Model

A steady-state transport analysis requires the definition of streamlines. The streamlines are the trajectories that the material follows during transport through the mesh. To meet this requirement, the mesh must be generated using the symmetric model generation capability, which is described in detail in *Symmetric Model Generation*. The three-dimensional model can be created either by revolving an axisymmetric model about its axis of revolution or by revolving a single three-dimensional repetitive sector about its axis of symmetry.

Revolving an Axisymmetric Cross-Section to Create a Three-Dimensional Model

You can generate a three-dimensional mesh by revolving a two-dimensional cross-section about a symmetry axis, so that the streamlines follow the mesh lines. In this case the symmetric model generation capability requires a two-dimensional cross-section of the body as a starting point. The cross-section, which must be discretized with axisymmetric finite elements, is defined in a separate input file. A data check analysis must be performed to write the model information to a restart file. The restart file is read in a subsequent run, and a three-dimensional model is generated by Abaqus/Standard by revolving the cross-section about the symmetry axis, starting at a reference plane. Both the symmetry axis and reference plane of the new three-dimensional model can be oriented in any direction in the global coordinate system. The symmetry axis also defines the axis of the spinning body. A nonuniform discretization in the circumferential direction can be specified to allow a finer mesh in the contact region than elsewhere in the model.

Input File Usage: **SYMMETRIC MODEL GENERATION*, REVOLVE

Revolving a Single Three-Dimensional Sector to Create a Periodic Model

Alternatively, you can generate a periodic three-dimensional mesh by revolving a single three-dimensional sector about its axis of symmetry. To accurately account for the material convection when the streamline integration is performed, the segment angle for the repetitive three-dimensional sector must be chosen small enough.

In this case the symmetric model generation capability requires a single three-dimensional sector as a starting point. The original three-dimensional sector is defined in a separate input file. A data check analysis must be performed to write the model information to a restart file. The restart file is read in a subsequent run, and a three-dimensional periodic model is generated by Abaqus/Standard by revolving the original three-dimensional sector about the symmetry axis. Both the symmetry axis and the original three-dimensional repetitive sector can be oriented in any direction in the global coordinate system. The symmetry axis also defines the axis of the spinning body. There is no restriction that the meshes on the two symmetry surfaces of the repetitive sector match in any way. If the surface meshes on either side of the original sector are not matched completely, constraints will be generated automatically to couple the opposing neighboring surfaces when revolving the original sector to create a periodic model.

Input File Usage: **SYMMETRIC MODEL GENERATION*, PERIODIC

Identifying the Elements Being Treated in an Eulerian Manner

By default, the rigid body motion in the whole model will be described in a spatial or Eulerian manner. In some cases you may want only part of the model to be treated with the Eulerian method while the rest should be treated with the classical Lagrangian method. One typical example is a disc brake where the disc itself can be treated with the Eulerian method while the brake assembly (brake pads and caliper) is treated with the Lagrangian method. In this case you can specify the name of an element set for which the rigid body motion will be described in an Eulerian manner. The elements that are not included in the element set will be treated with the classical Lagrangian method. Only one Eulerian element set can be specified in the whole model. In a new steady-state transport step or upon restart (see *Restarting an Analysis*) you can respecify a set of elements to be treated with the Eulerian method even after it has previously been treated with the Lagrangian method and vice versa. Elements treated with the Eulerian method and elements treated with the Lagrangian method cannot be mixed along a streamline.

Input File Usage: **STEADY STATE TRANSPORT*, ELSET=*name*

Defining Reference Frame Motions

The deformable and rigid bodies can each be defined in their own moving reference frame in a steady-state rolling and sliding analysis. The motion of these reference frames can be defined quite generally and provides modeling of a spinning deformable body traveling along a straight line, or "cornering" or "precessing" around an axis such as shown in *Figure 1*. It is also possible to define reference frame motions for rigid bodies, including translations and rotations. The rigid body can be flat, convex, or concave, which allows for modeling of a deformable body in contact with a rotating drum, such as a tire rolling on a drum, or for modeling a tire mounted on a rigid rim.

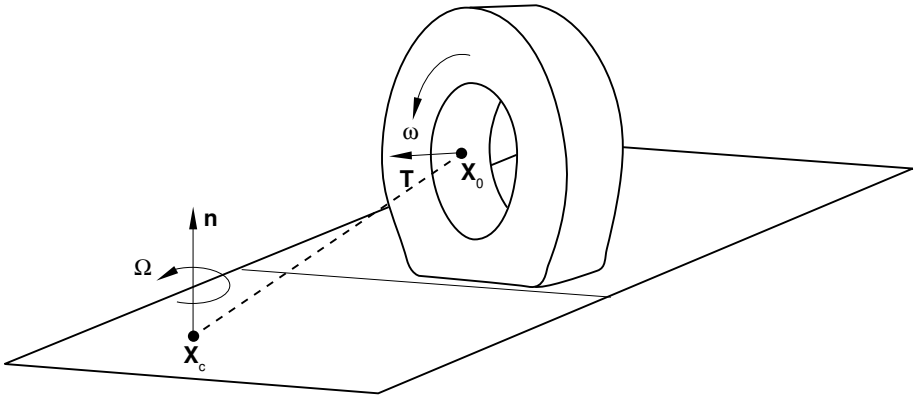


Figure 1: Constant cornering example showing conventions for defining reference frame motions.

When defining different reference frame motions for bodies that interact, you must make sure that the interactions are indeed steady. For example, for a planar rigid surface the relative reference frame motion must be tangential to the rigid surface, and for a body of revolution the relative reference frame motion must be rotation around its axis. Convergence difficulties will persist if the interactions are not steady.

Spinning Motion

The spinning motion of the deformable body around its own axis is described by a user-specified angular velocity, $\boldsymbol{\omega}$ (see *Figure 1*). This angular velocity defines the transport of material through the mesh; you define the magnitude of the spinning rotation, $\boldsymbol{\omega}$. The axis of revolution is the symmetry axis used for generating the mesh as described in *Defining the Model*. The transport velocity must be defined for all nodes on the spinning body. The magnitude of the angular velocity can also be defined with user subroutine *UMOTION*.

The transport velocity can also be applied to a rigid body based on a three-dimensional surface of revolution. In that case the velocity is applied to the rigid body reference node to describe the transport of the (rigid) material relative to the reference node. Abaqus/Standard assumes that the rigid body spins around the axis of revolution of the rigid body. This option can, for example, be applied to the rigid body representing the rim on which a tire is mounted.

Abaqus/Standard will automatically update the position and orientation of the rotation axis to the current configuration in a large-displacement analysis, such as in the case where a prescribed load applied to the reference node of a rotating rigid drum maintains the contact pressure between the tire and drum or the case where a camber angle is applied to the axle of the deformable body.

Input File Usage: Use either of the following options:

**TRANSPORT VELOCITY*
**TRANSPORT VELOCITY*, USER

Defining a Reference Frame for Translational or Rotational Motion

The rotating deformable body is also associated with a reference frame. This reference frame can either translate or rotate with respect to the fixed global reference frame. Similarly, each rigid body must be defined in a reference frame that is either fixed, translates, or rotates. For example, to associate straight line travel at ground velocity, $\mathbf{c}$, with a spinning deformable body, the deformable body can be defined in a reference frame translating at velocity $\mathbf{c}$ and the rigid surface can be defined in a fixed reference frame. Alternatively, the deformable body can be defined in a reference frame that does not translate and the rigid body can be defined in a frame translating at velocity $-\mathbf{c}$. Another example is a deformable body precessing along a circular path such as shown in *Figure 1*. In such a case a rotating frame is associated with the deformable body that defines the precession axis and angular velocity, while the rigid body is defined in a fixed reference frame. All components of the reference frame motion are zero unless otherwise specified; components of the reference frame motion cannot be treated as unknowns to be determined by the simulation.

You can apply a specified motion of the reference frame to all nodes of the deformable body or to the reference node of a rigid body. A translating reference frame is defined by specifying the components of the velocity vector, $\mathbf{c}$. A rotating reference frame is defined by specifying the magnitude of an angular rotation velocity, $\boldsymbol{\Omega}$, and the position and orientation of the axis of rotation in the current configuration. The position and orientation of the axis are applied at the beginning of the step and remain fixed during the step.

Input File Usage: Use the following option to define the motion of a translating reference frame:

**MOTION*, TRANSLATION

Use the following option to define the motion of a rotating reference frame:

**MOTION*, ROTATION

Contact Conditions

Abaqus/Standard provides contact between a rigid surface and deformable body moving with different velocities, such as contact between a rolling tire and the ground, as well as contact between surfaces moving with the same velocity, such as the contact between the bead and rim in a tire analysis. Abaqus/Standard also provides contact between two deformable bodies moving with the same velocity, such as the contact between the tread blocks on a tire surface, as well as contact between two deformable bodies moving with different velocities, such as the contact between a disc and brake assembly.

Contact between a Rigid Surface and a Deformable Body Moving with Different Velocities

The rigid surface can be either an analytical surface or made from rigid elements. When the main and secondary surfaces move with different velocities, you will normally select to use a Coulomb friction law that assumes that slip occurs if the frictional stress

$$\tau_{eq} = \sqrt{\tau_1^2 + \tau_2^2}$$

is equal to the critical stress $\tau_{crit} = \mu p$, where τ_1 and τ_2 are the shear stresses on the contact plane, μ is the friction coefficient, and p is the contact pressure. No slip occurs when $\tau_{eq} < \tau_{crit}$. For steady-state transport the condition of no slip is approximated in Abaqus/Standard by stiff "viscous" behavior

$$\tau_\alpha = \kappa_s \dot{\gamma}_\alpha,$$

where $\dot{\gamma}_\alpha$ are the tangential slip velocities that depend on deformation along a streamline and

$$\kappa_s = \frac{\mu p}{2F_f \omega R}$$

is the "stick viscosity," R is the radius of the cylinder, and F_f is a user-defined slip tolerance for which the default is 0.005. Using a larger slip tolerance makes convergence of the solution more rapid at the expense of solution accuracy. Using a smaller slip tolerance imposes the "no relative motion" constraint more accurately but may slow convergence. The default value provides a conservative balance between efficiency and accuracy for rolling contact problems.

Since this frictional model used for steady-state rolling is different from the frictional models used with other analysis procedures in Abaqus/Standard, discontinuities may arise in the solutions between a steady-state transport analysis and any other analysis procedure, such as a static footprint analysis. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to a steady-state rolling analysis use a zero coefficient of friction. You can then modify the friction properties in the steady-state transport analysis step to use the desired friction coefficient (see *Changing Friction Properties during an Abaqus/Standard Analysis*).

This frictional model is more relevant in a tire analysis since the velocity of the rotating tire strongly depends on the deformation gradients along a streamline on the contact surface. The solution state at a material point depends on the solution of neighboring points, and convective effects must be considered. However, since the deformation gradients along a streamline on the contact surface are small in a disc brake analysis, a simplified frictional model, which ignores the convective effect on the contact surface, can be used. Such a frictional model is discussed in the following section.

Contact between Two Deformable Bodies Moving with Different Velocities

When the secondary and main surfaces rotate with different velocities, such as contact between a disc and brake assembly, slip will develop between the two deformable surfaces. The transport velocity (*Spinning Motion*) and the motion of a reference frame (*Defining a Reference Frame for Translational or Rotational Motion*) can be defined in a steady-state transport analysis procedure to model the steady-state frictional sliding between two deformable bodies that are moving with different velocities. In this case it is assumed that the slip rate simply follows from the difference in velocities specified by the transport velocity and the motion of the reference frame and is independent of the deformation gradient along a streamline or the nodal displacements on the contact surface. No convective effects are considered between the contact surfaces, and the frictional stress does not depend on any history effects. Hence, the frictional stress is given by

$$\tau_\alpha = \mu p \frac{\dot{\gamma}_\alpha}{|\dot{\gamma}_\alpha|} = \mu p t_\alpha,$$

where μ is the friction coefficient, p is the contact pressure, t_α are the local tangent directions, and $\dot{\gamma}_\alpha$ are the slip velocities that are defined by the transport velocity and the motion of the reference frame. If no velocity or the same velocity are defined at contact nodes with friction, sticking conditions are applied automatically. The friction model is described in detail in *Coulomb friction*.

Such a simplified frictional model is relevant only in a disc brake analysis. It should be used with care in a rolling tire analysis where deformation gradients on the contact surface are significant.

Since this frictional behavior is different from the frictional models used with other analysis procedures in Abaqus/Standard, discontinuities may arise in the solutions between a steady-state transport analysis and any other analysis procedure. An example is the discontinuity that occurs between the initial preloading of the disc pads in a disc brake system and the subsequent braking analysis where the disc spins with a prescribed rotation. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to a steady-state analysis use a zero coefficient of friction (see *Including Friction Properties in a Contact Property Definition*). You can then increase the friction coefficient to the desired value in the steady-state transport analysis (see *Changing Friction Properties during an Abaqus/Standard Analysis*).

Contact between Surfaces Spinning with the Same Angular Velocity

When the secondary and main surfaces rotate with the same angular velocity, such as the surface between the bead and rim in a tire analysis, no relative velocity develops between the surfaces. In such a case, frictional stresses develop as a reaction between the bodies. Abaqus/Standard will automatically determine that the secondary and main surface rotate with the same speed and apply the standard Coulomb friction model, which is described in detail in *Frictional Behavior*.

When the standard Coulomb friction model is used in a reference frame that implies flow of material through the mesh, convective effects must be considered. However, Abaqus/Standard assumes that no convective effects are present between surfaces during steady-state transport analysis. In other words, Abaqus/Standard assumes that the frictional stress at a point depends on the history of deformation in the Lagrangian reference frame and ignores any history effects that may occur as a result of the deformation that the point experiences during the spinning motion. The assumption that the frictional stress does not depend on history effects during rolling is valid for modeling contact between a tire bead and rim where relative slip occurs only during rim mounting in a static analysis prior to the steady-state transport analysis. When slip occurs during the steady-state transport analysis, the solution obtained is no longer the correct steady-state solution because convective effects are ignored. To ensure that no slip takes place between the surfaces during steady-state rolling, it is recommended that you modify the friction properties in the steady-state transport analysis step to activate rough friction (see *Changing Friction Properties during an Abaqus/Standard Analysis*).

Incrementation

Abaqus/Standard uses Newton's method to solve the nonlinear equilibrium equations. The nonlinearities in a steady-state transport analysis arise from large-displacement effects, material nonlinearity, and boundary nonlinearities such as contact and friction. If geometrically nonlinear behavior is expected other than the large rigid body rotation associated with the steady-state motion, the step definition should include nonlinear geometric effects.

The steady-state rolling and sliding solution must often be obtained as a series of increments, with iterations to obtain equilibrium within each increment. If the direct steady-state solution technique is used, the solution in each increment is a steady-state solution corresponding to the loads acting on the structure at that instant. If the pass-by-pass steady-state solution technique is used, the solution in each increment is usually not a steady-state solution corresponding to the loads acting on the structure at that instant. In this case a steady-state solution is generally obtained in several increments, with each increment corresponding to a loading pass through the structure.

Since Newton's method has a finite radius of convergence, too large an increment in the applied load can prevent any solution from being obtained because the current steady-state solution is too far away from the new steady-state equilibrium solution that is being sought: it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic Incrementation

In most cases the default automatic incrementation scheme is preferred because it will select increment sizes based on computational efficiency.

Input File Usage: **STEADY STATE TRANSPORT*

Direct Incrementation

Direct user control of the increment size is also provided because if you have considerable experience with a particular problem, you may be able to select a more economical approach.

Input File Usage: **STEADY STATE TRANSPORT*, DIRECT

Using the Maximum Number of Iterations to Determine the Increment Size

The solution to an increment can be accepted after the maximum number of iterations allowed has been completed (as defined in *Commonly Used Control Parameters*), even if the equilibrium tolerances are not satisfied. This approach is not recommended; it should be used only in special cases when you have a thorough understanding of how to interpret results obtained in this way. Very small increments and a minimum of two iterations are usually necessary in this case.

Input File Usage: **STEADY STATE TRANSPORT*, DIRECT=NO STOP

Convergence in a Steady-State Transport Analysis

The steady-state transport procedure may experience convergence difficulties in certain situations that are described below.

Convergence Issues with Friction

The frictional forces that develop on the contact surface as a result of steady-state rolling are functions of the spinning angular velocity, $\boldsymbol{\omega}$, and the traveling straight line velocity, $\mathbf{c}$, or cornering velocity, $\boldsymbol{\Omega}$. When these frictional forces are large, convergence of Newton's method becomes difficult. Convergence problems in Abaqus/Standard are usually resolved by taking a smaller load increment. However, contact forces due to steady-state rolling usually do not reduce when the magnitudes of the velocities are reduced. For example, if a spinning object is prevented from moving ($\mathbf{c} = \mathbf{0.0}$), full slipping conditions will develop over the entire contact zone for all values of spinning angular velocity $\boldsymbol{\omega} > \mathbf{0.0}$. Consequently, the frictional force remains constant for all $\boldsymbol{\omega} > \mathbf{0.0}$ (provided that the normal force remains constant), so that smaller increments in the velocities ($\boldsymbol{\omega}, \boldsymbol{\Omega}, \mathbf{c}$) do not reduce the magnitude of the frictional forces and, hence, do not overcome convergence difficulties.

To provide for convergence through the use of smaller increments in such cases, the friction coefficient can be increased from zero to the desired value over the analysis step. This is accomplished by setting the initial friction coefficient for the model to zero (see *Including Friction Properties in a Contact Property Definition*), then increasing the friction coefficient to its final value in the steady-state transport analysis step (see *Changing Friction Properties during an Abaqus/Standard Analysis*).

Stiffness terms for Newton iterations associated with steady-state transport rely on derivatives numerically computed along streamlines. These computations can be sensitive to mesh discretization details and whether a surface-to-surface or node-to-surface contact formulation is used. Sensitivity in convergence behavior depending on which contact formulation is used will tend to diminish upon mesh refinement. (See *Contact Formulations in Abaqus/Standard* for further discussion of contact formulations).

Convergence Issues with Inertial Loads

At higher rotational velocities, inertia forces can give rise to instabilities in the form of standing waves, which are likely to prevent convergence of the Newton algorithm.

Unsymmetric Inertia Load Stiffness Matrix

If unsymmetric matrix storage is active during a steady-state transport step, unsymmetric inertia load stiffness matrix terms are computed for axisymmetric three-dimensional models (models created by revolving an axisymmetric mesh cross-section during symmetric model generation). At high rolling speeds the unsymmetric inertia load stiffness terms can improve the convergence behavior of the simulation significantly. However, these

terms involve nonlocal element calculations, which increases the equation solver bandwidth and, consequently, the computation expense. At low rolling speeds the inertia load stiffness terms are often not critical for convergence and can be ignored to shorten the simulation time.

Input File Usage: Use the following option to include inertia forces but ignore unsymmetric inertia load stiffness matrix terms:

**STEADY STATE TRANSPORT*, INERTIA=LOW SPEED

Inertia Stabilization

In addition to the unsymmetric inertia load stiffness matrix terms, Abaqus also provides a form of upwind stabilization (referred to as inertia stabilization) to improve convergence at high speeds. This algorithm is available only for three-dimensional axisymmetric models meshed with first-order elements, with the exception of C3D8I elements.

Inertia stabilization is numerical damping that applies an artificial resultant force and moment on the model. In general, you should set the stabilization factor to the smallest value that still provides reasonable convergence. Typically, the default inertia stabilization factor of one provides noticeably improved convergence at high speeds without negatively impacting the results. Nodal output variables are available to obtain the force and moment due to inertia stabilization and can be compared against the total force and moment due to inertia effects to assess the effect of artificial stabilization (see *Output*). You can turn off inertia stabilization by setting the inertia stabilization factor to zero.

Input File Usage: Use the following option to control the amount of inertia stabilization to apply:

**STEADY STATE TRANSPORT*, INERTIA STABILIZATION FACTOR=*value*

Convergence Issues with the Mullins Effect Material Model

If the Mullins effect material model is included in the material definition (see *Stress Softening in Rubberlike Materials*), there could be a strong discontinuity in the response of a structure in transitioning from a static (non-rolling) state to a steady-state rolling state. This discontinuity is due to the damage that occurs during the transient response (such as the damage that occurs as the structure undergoes its first revolution after static preloading). Since the transient response is not modeled during a steady-state transport analysis, the resulting discontinuity in the response can lead to convergence problems. The damage associated with the Mullins effect is independent of the angular speed of rotation: as a result, time increment cutbacks do not resolve the convergence problems. The Mullins effect can be ramped up over the time period of the step in these situations to obtain a converged solution. In such a case the change in response due to damage is applied gradually over the step. The solution at the end of the step corresponds to the fully damaged material; solutions during the step correspond to a partially damaged material and are, therefore, physically meaningless. Thus, it is recommended that in going from a static to a steady-state rolling solution, a do-nothing step at a low angular speed of rotation be first carried out with the Mullins effect ramped on. This facilitates resolution of the discontinuity in a gradual manner. The do-nothing step can then be followed by the regular steady-state transport step with the Mullins effect applied instantaneously at the beginning of the step. This approach is illustrated in *Analysis of a solid disc with Mullins effect and permanent set*.

Input File Usage: **STEADY STATE TRANSPORT*, MULLINS=RAMP *or* STEP (default)

Convergence Issues with Streamline Integration in Plasticity/Creep Models

Although in principle any material point along a streamline can be used as a starting point for the streamline integration when material convective calculations are performed, Abaqus/Standard always uses the material

points in the original sector or the material points in the original cross-section as starting points for the streamline integration in a model with periodic geometry or axisymmetric geometry, respectively.

If the pass-by-pass solution technique is used, after an increment has been performed for all the streamlines, Abaqus/Standard will automatically use the state obtained at the end of the streamline as the starting state for the streamline integration in the subsequent increment. This iterative process is repeated for each increment until a steady-state solution is reached.

If the direct steady-state solution technique is used, several local iterations are usually required for each streamline, with a local iteration corresponding to an integration over a closed loop streamline. After a local iteration has been performed for a streamline, Abaqus/Standard will check to see if the steady-state condition is satisfied for the streamline. This is best measured by ensuring the differences between the stresses/strains at the starting point of the streamline obtained before and after the iteration are sufficiently small. If the steady-state condition is not satisfied for the streamline, Abaqus/Standard will automatically use the state obtained at the end of the previous local iteration as the starting state for the streamline integration in the subsequent local iteration. This iterative process is repeated until a steady-state solution is reached for all the streamlines.

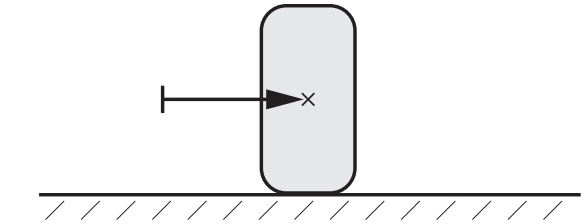
To improve the rate of convergence, it is recommended that you apply loads on elements or nodes away from the starting points of the streamlines.

Convergence Issues with Unconstrained Mesh Motion

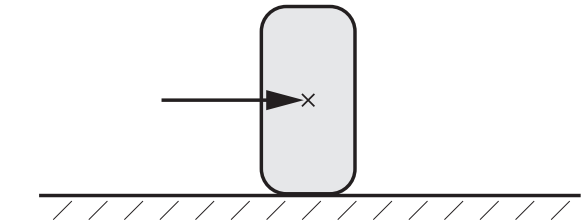
Unconstrained rigid body modes of the mesh motion will cause convergence problems for a steady-state transport analysis, similar to convergence problems for unconstrained rigid body modes in a static analysis. Friction cannot be relied on to restrict rigid body modes in a steady-state transport analysis, because frictional stresses depend on relative material velocities rather than relative nodal displacements for steady-state transport. Restricting the (steady-state) material velocity does not restrict nodal displacements for steady-state transport analyses. The material velocity includes effects of material flowing through the mesh and is governed by the spinning motion (see *Spinning Motion*), reference frame motions (see *Defining a Reference Frame for Translational or Rotational Motion*), and nodal positions relative to the spinning axis.

Consider the examples shown in *Figure 2*. End-on views are shown in *Figure 2*, so the axial direction (spinning axis) is horizontal in this figure. The axial component of the reference frame motion is zero for all three of these cases (either by explicit specification or implicitly by default). Since the material velocity in the axial direction at steady state is zero for both bodies (according to the reference frame motion), the frictional force in the axial direction will remain zero for all three of these cases, which may not be intuitive. The first case shown in *Figure 2* (a) involves a planar interface and a boundary condition in the axial direction. At the time the rolling body reaches steady state, the axial motion has stopped, the axial friction force is zero, and the reaction force associated with the boundary condition is zero (which also may not be intuitive).

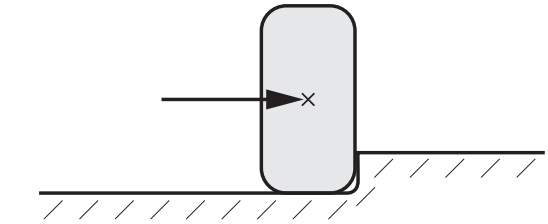
The second case in *Figure 2* (b) has a planar interface and an applied force in the axial direction. The axial frictional force for the steady-state solution remains zero, as already discussed, so no axial force arises to counter the applied force. Therefore, Abaqus/Standard will not provide a converged solution in this case, which is an example of unconstrained rigid body mesh motion. The third case shown in *Figure 2* (c) is like the second case, except a “curb” has been added to the rigid surface. In this case, a contact force in the normal direction occurs at the location of the “curb,” which counters the applied force, so the analysis is able to converge.



a) Prescribed tangential displacement produces no reaction force.



b) Concentrated force makes the problem unsteady.



c) Due to curved interface the wheel can withstand concentrated force.

Figure 2: Displacement boundary condition versus concentrated force.

As a real world example, consider a car traveling along a straight planar road with a truck moving parallel to the car, applying a constant concentrated force that pushes the car sideways. With zero toe angle on the car's front wheels (that is, the wheels are exactly aligned with the longitudinal axis of the car), steady-state motion is impossible, and the car will eventually slip off the road. To resist the push in steady-state motion, the car wheels need to be aligned with the proper toe angle.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified. *Initial Conditions* describes all of the available initial conditions.

Boundary Conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6). (See *Boundary Conditions* for details of applying boundary conditions to rotation degrees of freedom when large rotation will occur.) During the analysis prescribed boundary conditions can be varied using an amplitude definition (see *Amplitude Curves*). Boundary conditions restrict the mesh motion but do not restrict the transport of material through the mesh (due to the spinning motion discussed in *Spinning Motion*).

Loads

Loading in a steady-state transport analysis includes the motion of the structure, inertia (d'Alembert) forces due to motion, concentrated loads, distributed pressures, and body forces.

Inertia Effects

The motion of the deformable body gives rise to inertia (d'Alembert) forces that can be included. These forces include centrifugal and Coriolis effects. The density of the material must be defined in the material description.

Abaqus automatically computes unsymmetric inertia load stiffness matrix terms for axisymmetric three-dimensional models if unsymmetric matrix storage is active. These terms are essential to obtain converged solutions at high rolling speeds (see *Unsymmetric Inertia Load Stiffness Matrix*). However, at low rolling speeds, the inertia load stiffness terms are often not critical for convergence and can be ignored to reduce computation effort.

Input File Usage: Use the following option to include inertia forces:

**STEADY STATE TRANSPORT*, INERTIA=YES

Use the following option to include inertia forces but ignore unsymmetric inertia load stiffness matrix terms:

**STEADY STATE TRANSPORT*, INERTIA=LOW SPEED

Inertia Loads for Tetrahedral Elements

Inertia loads for tetrahedral elements C3D4, C3D10, C3D10HS, and C3D10M are not taken into account in a steady-state transport analysis. Tetrahedral elements will appear only in a periodic model created by revolving a three-dimensional sector that contains tetrahedral elements. Tetrahedral elements will not appear in an axisymmetric model created by revolving a two-dimensional cross-section about a symmetry axis. See *Symmetric Model Generation* for details.

Other Prescribed Loads

The following loads can be prescribed in a steady-state transport analysis, as described in *Concentrated Loads*:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in *Abaqus Elements Guide*.

In most cases such loads should be applied around the whole circumference of the body; a load on a single point or element corresponds to a spatially fixed load, which in most cases is not realistic.

Predefined Fields

The following predefined fields can be specified in a steady-state transport analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in a steady-state transport analysis, nodal temperatures can be specified as a predefined field. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.

Material Options

Since the steady-state transport capability uses a kinematic description that implies flow of material through the mesh, convective effects must be considered for the material response. Most material models that describe mechanical behavior (except user-defined materials) are available for use in a steady-state transport analysis. In particular, history-dependent viscoelasticity (*Time Domain Viscoelasticity*), history-dependent Mullins effect (*Stress Softening in Rubberlike Materials*), classical metal plasticity (*Classical Metal Plasticity*), rate-dependent yield (*Rate-Dependent Yield*), rate-dependent creep (*Rate-Dependent Plasticity: Creep and Swelling*), two-layer viscoplasticity (*Two-Layer Viscoplasticity*), and models defined using the parallel rheological framework (*Parallel Rheological Framework*) can all be used during a steady-state transport analysis. However, models defined within the parallel rheological framework cannot include plasticity.

The following material properties are not active during a steady-state transport analysis: thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties.

Abaqus/Standard also provides the ability to obtain the fully relaxed long-term elastic or elastic-plastic solution during a steady-state transport analysis if the material description includes viscoelastic or viscoplastic material properties. If the material description includes viscoelastic material properties, the long-term solution will ignore the material convection calculations. If the two-layer viscoplastic material model is used, the long-term solution will include only the material convection calculations based on the long-term response of the elastic-plastic network.

Input File Usage: **STEADY STATE TRANSPORT*, LONG TERM

Choosing an Appropriate Material Model

Since material points in a spinning and sliding body undergo repeated loading/unloading cycles, an appropriate material model must be chosen to characterize the response correctly under such loading conditions. The use of plasticity material models with isotropic type hardening is generally not recommended since they will continue to harden during cyclic loading, which may lead to a large number of iterations until the steady-state solution is reached. Kinematic hardening plasticity models should be used to model the inelastic behavior of materials that are subjected to repeated loading.

For rate-dependent creep, the two-layer viscoplasticity model is recommended (*Two-Layer Viscoplasticity*) for modeling the response of materials with significant time-dependent behavior as well as plasticity at elevated temperatures.

For history-dependent viscoelasticity, it is more appropriate to use cyclic (frequency domain) test data to calibrate the time-domain viscoelastic material model for steady-state transport analysis. The cyclic experiments should be performed in the frequency range anticipated in the rolling simulation. Abaqus/Standard internally converts the frequency domain storage and loss modulus data into a time-domain (Prony series) representation. This data conversion capability is described in detail in *Time Domain Viscoelasticity*.

Analysis Steps prior to a Steady-State Transport Analysis

It is recommended that the solutions in any analysis step prior to a steady-state transport analysis, such as a static footprint or preloading solution, be based on the long-term elastic moduli or the long-term elastic-plastic response if viscoelastic or viscoplastic material properties are used (for example, see *Static Stress Analysis*). The long-term solution provides a smooth transition between a static analysis and a slow rolling or sliding steady-state transport analysis.

Material Convection in Nonlinear Analysis

When material convection is included in the steady-state transport solution, Abaqus/Standard uses an approximate Jacobian matrix in the Newton solution of the nonlinear equilibrium equations. The rate of convergence in such a case is no longer quadratic but depends strongly on the severity of the nonlinearities. It is often necessary to adjust the default solution controls (*Commonly Used Control Parameters*) to obtain a steady-state transport solution when material convection is considered.

Elements

Most of the three-dimensional stress/displacement elements in Abaqus/Standard can be used in a steady-state transport analysis (see *Choosing the Appropriate Element for an Analysis Type*). When the three-dimensional model is generated from an axisymmetric cross-section, the element type used in the two-dimensional model determines the element type in the three-dimensional model. The correspondence between the two-dimensional and three-dimensional element types is described in *Symmetric Model Generation*. If the three-dimensional periodic model is generated from a single three-dimensional sector, any of the stress/displacement elements in Abaqus/Standard can be used.

Output

The element output available for a steady-state transport analysis includes stress, strain, energies, and the values of state, field, and user-defined variables. The nodal output available includes displacements, velocities, reaction forces, and coordinates. The contact output variable CSLIP contains steady-state slip rates for the steady-state transport procedure, unlike the usual definition of this variable. All of the output variable identifiers are outlined in *Using Abaqus/Standard Output Variable Identifiers*.

In addition to the usual output variables available in Abaqus/Standard, the following variables are provided specifically for steady-state transport analysis:

SSTIF
Nodal forces due to steady-state transport inertia loading. This output allows you to visualize the inertial forces generated in a deformable spinning wheel.

SSTSF
Nodal forces due to inertia stabilization in a steady-state transport analysis. This output allows you to visualize the artificial inertia stabilization loads and compare them to the inertial forces that output variable SSTIF generated in a deformable spinning wheel.

SSTIRF
Resultant of all inertia nodal loads in a steady-state transport analysis. The node set prescribed for this history output request must have only one member. Because the output is a resultant of forces, the node chosen for output does not affect the computation. Typically, output is requested on a reference node on the axle of a wheel.

SSTSRF
Resultant of all inertia stabilization loads in a steady-state transport analysis. The node set prescribed for this history output request must have only one member. Because the output is a resultant of forces, the node chosen for output does not affect the computation. Typically, output is requested on a reference node on the axle of a wheel. This output allows you to compare the relative magnitudes of the resultant inertia nodal loads (output variable SSTIRF) to that of the artificial inertia stabilization loads.

SSTIRM
Resultant moment of all inertia nodal loads in a steady-state transport analysis. The node set prescribed for this history output request must have only one member. Because the output is a

resultant moment, the node chosen for output does affect the computation. Typically, output is requested on a reference node on the axle of a wheel.

SSTSRM

Resultant moment of all inertia stabilization loads in a steady-state transport analysis. The node set prescribed for this history output request must have only one member. Because the output is a resultant moment, the node chosen for output does affect the computation. Typically, output is requested on a reference node on the axle of a wheel. This output allows you to compare the relative magnitudes of the resultant moment due to inertia nodal loads (output variable SSTIRM) to that of the artificial inertia stabilization loads. Understanding this value is particularly important while looking for a free-rolling solution of a tire in which a zero reaction moment about the axle is sought.

Limitations

The steady-state transport analysis capability has several limitations.

- The deformable structure must be a full 360° cylindrical body of revolution. Convective boundary conditions are not available to model segments of a cylinder.
- The capability is not available in two dimensions.
- Only one deformable spinning body is permitted. The symmetric model generation capability must be used to generate the deformable body (*Symmetric Model Generation*).
- Material model definitions for models defined using the parallel rheological framework (*Parallel Rheological Framework*) cannot include plasticity.

Input File Template

```
*HEADING
…
*SYMMETRIC MODEL GENERATION, REVOLVE
Data lines to define model generation
*SURFACE INTERACTION
*FRICTION
Specify zero friction coefficient
**
*STEP
*STATIC
Data lines to define analysis steps prior to transport analysis
*END STEP
…
*STEP
*STEADY STATE TRANSPORT
Data line to define incrementation
*CHANGE FRICTION
*FRICTION
Data lines to redefine friction coefficient
*BOUNDARY
Data lines to define boundary conditions
*TRANSPORT VELOCITY
Data lines to define spinning angular velocity
*MOTION, TRANSLATION or ROTATION
Data lines to define traveling velocity or cornering rotational velocity
*EL PRINT and/or *NODE PRINT
```

Data lines to request output variables
```
*END STEP
```

Heat Transfer and Thermal-Stress Analysis

In this section:

- *About Heat Transfer Analysis Procedures*
- *Uncoupled Heat Transfer Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Adiabatic Analysis*

About Heat Transfer Analysis Procedures

Overview

Abaqus can solve several types of heat transfer problems.

Abaqus can solve the following types of heat transfer problems:

Uncoupled heat transfer analysis

Heat transfer problems involving conduction, forced convection, and boundary radiation can be analyzed in Abaqus/Standard. See *Uncoupled Heat Transfer Analysis*. In these analyses the temperature field is calculated without knowledge of the stress/deformation state or the electrical field in the bodies being studied. Pure heat transfer problems can be transient or steady-state and linear or nonlinear.

Sequentially coupled thermal-stress analysis

If the stress/displacement solution is dependent on a temperature field but there is no inverse dependency, a sequentially coupled thermal-stress analysis can be conducted in Abaqus/Standard. Sequentially coupled thermal-stress analysis is performed by first solving the pure heat transfer problem, then reading the temperature solution into a stress analysis as a predefined field. See *Sequentially Coupled Thermal-Stress Analysis*. In the stress analysis the temperature can vary with time and position but is not changed by the stress analysis solution. Abaqus allows for dissimilar meshes between the heat transfer analysis model and the thermal-stress analysis model. Temperature values will be interpolated based on element interpolators evaluated at nodes of the thermal-stress model.

Fully coupled thermal-stress analysis

A coupled temperature-displacement procedure is used to solve simultaneously for the stress/displacement and the temperature fields. A coupled analysis is used when the thermal and mechanical solutions affect each other strongly. For example, in rapid metalworking problems the inelastic deformation of the material causes heating, and in contact problems the heat conducted across gaps may depend strongly on the gap clearance or pressure.

Both Abaqus/Standard and Abaqus/Explicit provide coupled temperature-displacement analysis procedures, but the algorithms used by each program differ considerably. In Abaqus/Standard the heat transfer equations are integrated using a backward-difference scheme, and the coupled system is solved using Newton's method. These problems can be transient or steady-state and linear or nonlinear. In Abaqus/Explicit the heat transfer equations are integrated using an explicit forward-difference time integration rule, and the mechanical solution response is obtained using an explicit central-difference integration rule. Fully coupled thermal-stress analysis in Abaqus/Explicit is always transient. Cavity radiation effects cannot be included in a fully coupled thermal-stress analysis. See *Fully Coupled Thermal-Stress Analysis* for more details.

Fully coupled thermal-electrical-structural analysis

A coupled thermal-electrical-structural procedure is used to solve simultaneously for the stress/displacement, the electrical potential, and the temperature fields. A coupled analysis is used when the thermal, electrical, and mechanical solutions affect each other strongly. An example of such a process is resistance spot welding, where two or more metal parts are joined by fusion at discrete points at the material interface. The fusion is caused by heat generated due to the current flow at the contact points, which depends on the pressure applied at these points.

These problems can be transient or steady-state and linear or nonlinear. Cavity radiation effects cannot be included in a fully coupled thermal-electrical-structural analysis. This procedure is available only in Abaqus/Standard. See *Fully Coupled Thermal-Electrical-Structural Analysis* for more details.

Adiabatic analysis

An adiabatic mechanical analysis can be used in cases where mechanical deformation causes heating, but the event is so rapid that this heat has no time to diffuse through the material. Adiabatic analysis can be performed in Abaqus/Standard or Abaqus/Explicit; see *Adiabatic Analysis*. An adiabatic analysis can be static or dynamic and linear or nonlinear.

Coupled thermal-electrical analysis

A fully coupled thermal-electrical analysis capability is provided in Abaqus/Standard for problems where heat is generated due to the flow of electrical current through a conductor. See *Coupled Thermal-Electrical Analysis*.

Cavity radiation

In Abaqus/Standard cavity radiation effects can be included (in addition to prescribed boundary radiation) in uncoupled heat transfer problems. See *Cavity Radiation in Abaqus/Standard*. The cavities can be open or closed. Symmetries and blocking within cavities can be modeled. view factors are calculated automatically, and motion of objects bounding a cavity can be prescribed during the analysis. Cavity radiation problems are nonlinear and can be transient or steady-state.

References:

Uncoupled Heat Transfer Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

Uncoupled heat transfer problems:

- are those in which the temperature field is calculated without consideration of the stress/deformation or the electrical field in the bodies being studied;
- can include conduction, boundary convection, and boundary radiation;
- can be transient or steady-state; and
- can be linear or nonlinear.

In Abaqus/Standard uncoupled heat transfer problems:

- involve heat transfer in solid bodies;
- can include cavity radiation effects—see *Cavity Radiation in Abaqus/Standard*;
- can include forced convection through the mesh if forced convection/diffusion heat transfer elements are used;
- can include thermal interactions such as gap radiation, conductance, and heat generation between contact surfaces—see *Thermal Contact Properties*;
- can include thermal material behavior defined in user subroutine *UMATHT*—see *User-Defined Thermal Material Behavior*; and
- require the use of heat transfer elements.

References:

- *Defining an Analysis*
- *About Heat Transfer Analysis Procedures*
- **HEAT TRANSFER*
- *Including volumetric heat flux due to internal heat generation*
- *Configuring a heat transfer procedure*

Heat Transfer Analysis in Abaqus/Standard

Uncoupled heat transfer analysis is used to model solid body heat conduction with general, temperature-dependent conductivity, internal energy (including latent heat effects), and quite general convection and radiation boundary conditions, including cavity radiation. Forced convection of a fluid through the mesh can be modeled by using forced convection/diffusion elements.

Sources of Nonlinearity in a Heat Transfer Analysis

Heat transfer problems can be nonlinear because the material properties are temperature dependent or because the boundary conditions are nonlinear. Usually the nonlinearity associated with temperature-dependent material properties is mild because the properties do not change rapidly with temperature. However, when latent heat effects are included, the analysis may be severely nonlinear (see *Latent Heat*).

Boundary conditions are very often nonlinear; for example, film coefficients can be functions of surface temperature. Again, the nonlinearities are often mild and cause little difficulty. An exception is the "boiling" film condition, in which the film coefficient can change very rapidly because the fluid adjacent to the surface boils. A rapidly changing film condition (within a step or from one step to another) can be modeled easily using temperature-dependent and field-variable-dependent film coefficients. Radiation effects always make heat transfer problems nonlinear. Nonlinearities in radiation grow as temperatures increase.

Abaqus/Standard uses an iterative scheme to solve nonlinear heat transfer problems. The scheme uses the Newton method with some modifications to improve stability of the iteration process in the presence of highly nonlinear latent heat effects.

Steady-state cases involving severe nonlinearities are sometimes more effectively solved as transient cases because of the stabilizing influence of the heat capacity terms. The required steady-state solution can be obtained as the very long transient time response; the transient will simply stabilize the solution for that long time response.

Matrix Storage and Solution Scheme

In heat transfer analyses involving cavity radiation or forced convection/diffusion elements, the system of equations is unsymmetric. The nonsymmetric matrix storage and solution scheme is invoked automatically in these cases (see *Defining an Analysis*).

Steady-State Analysis

Steady-state analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. The problem then has no intrinsic physically meaningful time scale. Nevertheless, you can assign an initial time increment, a total time period, and maximum and minimum allowed time increments to the analysis step, which is often convenient for output identification and for specifying prescribed temperatures and fluxes with varying magnitudes.

Any fluxes or boundary condition changes to be applied during a steady-state heat transfer step should be given within the step, using appropriate amplitude references to specify their "time" variations (*Amplitude Curves*). If fluxes and boundary conditions are specified for the step without amplitude references, they are assumed to change linearly with "time" during the step, from their magnitudes at the end of the previous step (or zero, if this is the beginning of the analysis) to their newly specified magnitudes at the end of the heat transfer step.

Input File Usage: **HEAT TRANSFER*, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Steady state**

Automatic Incrementation

When steady-state analysis is chosen, you suggest an initial "time" increment and define a "time" period for the step; Abaqus/Standard then increments through the step accordingly. By default, Abaqus/Standard automatically determines a suitable increment size for each increment of the step.

Fixed Incrementation

You can also use a fixed incrementation scheme, in which Abaqus/Standard uses the same increment size for the duration of the step. The suggested initial "time" increment, Δt_0, defines the increment size.

Input File Usage: Set the initial increment, minimum increment size, and maximum increment size to the same value:

*HEAT TRANSFER, STEADY STATE
Δt_0, , Δt_0, Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Steady-state**: **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Transient Analysis

Time integration in transient problems is done with the backward Euler method (sometimes also referred to as the modified Crank-Nicholson operator) in the pure conduction elements. This method is unconditionally stable for linear problems.

The forced convection/diffusion elements use the trapezoidal rule for time integration. They include numerical diffusion control (the "upwinding" Petrov-Galerkin method) and, optionally, numerical dispersion control. The elements with dispersion control offer improved solution accuracy in cases where the transient response of the fluid is important. Artificial dispersion control introduces a stability limit on the size of the time increment such that the local Courant number

$$C = |\mathbf{v}| \frac{\Delta t}{\Delta \ell}$$

must be less than 1, where Δt is the time increment, $|\mathbf{v}|$ is the magnitude of the velocity vector, and $\Delta \ell$ is a characteristic element length in the direction of flow; that is, heat cannot be convected across more than one element length, $\Delta \ell$, in a single increment of time. In a uniform velocity field, the smallest element will dictate the stable time increment. Approximate calculation of the Courant number, C, is helpful during the mesh design stages so that excessively small stable time increments can be avoided. The elements without dispersion control have no such stability limit; therefore, it may be more economical to use the elements without this feature in transient cases where transient effects in the fluid itself are not a critical part of the solution (for example, when the important solution is the temperature field in the solid bodies that are included in the model, and when characteristic transient times in the fluid are very much shorter than characteristic transient times in the solids).

Time incrementation in a transient heat transfer analysis can be controlled directly by you or automatically by Abaqus/Standard. Automatic time incrementation is generally preferred.

Automatic Incrementation

The time increments can be selected automatically based on the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see *Time Integration Accuracy in Transient Problems*).

Input File Usage: *HEAT TRANSFER, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed Incrementation

If you select direct incrementation and do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will then be used throughout the analysis.

Input File Usage: **HEAT TRANSFER*
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious Oscillations due to Small Time Increments

In a transient heat transfer analysis with second-order elements, there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta \ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and might cause problems if temperature-dependent material properties are present. Abaqus/Standard provides no check on the user-defined initial time increment; you must ensure that the given value does not violate the above criterion.

In a transient analysies using first-order elements, Abaqus uses nodal integration of the heat capacity term by default, resulting in a lumped treatment for this term. This treatment eliminates the nonphysical oscillations but can lead to locally inaccurate solutions, especially in terms of the heat flux for small time increments. If smaller time increments are required, you should use a finer mesh in regions where the temperature changes occur. A nondefault option for integrating the heat capacity terms at the Gauss integration points is also available. Selection of this option might lead to nonphysical oscillations in the solution if the time increment violates the guideline provided above.

Unless you specify a maximum allowable time increment size as part of the heat transfer step definition, there is no upper limit on the time increment size (the integration procedure is unconditionally stable, at least for linear problems). However, if forced convection/diffusion elements including numerical dispersion control (element types DCC*xx*D) are included in the model, there is a numerical stability limit on the allowable time increment.

The requirement is that $\Delta t \leq \Delta \ell / |\mathbf{v}|$, where $|\mathbf{v}|$ is the magnitude of the fluid velocity and $\Delta \ell$ is a characteristic element length in the direction of flow. Abaqus/Standard will adjust the time increment automatically to satisfy this stability limit.

Ending a Transient Analysis

A transient analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or after the given time period, whichever comes first. Steady state is defined by the temperature change rate: when the temperature at every temperature degree of freedom changes at a rate that is less than the user-specified rate (given as part of the step definition), the analysis terminates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

**HEAT TRANSFER*, END=PERIOD (default)

Use the following option to end the analysis based on the temperature change rate:

**HEAT TRANSFER*, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **End step when temperature change is less than**

Internal Heat Generation

You can define volumetric heat generation within a material in user subroutine *HETVAL* or user subroutine *UMATHT*. You can use these user subroutines in the same analysis.

Defining Internal Heat Generation in User Subroutine `HETVAL`

If you define internal heat generation in user subroutine *HETVAL*, you must include heat generation in the material definition with the other thermal property definitions.

Heat generation might be associated with (relatively low) energy phase changes occurring during the solution. Such heat generation usually depends on state variables (such as the fraction transformed), which themselves evolve with the solution and are stored as solution-dependent state variables (see *About User Subroutines and Utilities*). The heat generation is computed in user subroutine *HETVAL*, where any associated state variables can also be updated. The subroutine will be called at all material calculation points for which the material definition includes heat generation.

Input File Usage: **HEAT GENERATION*

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Heat Generation**

Defining Internal Heat Generation in User Subroutine `UMATHT`

If you define internal heat generation in user subroutine *UMATHT*, you must also define the constitutive thermal behavior in the subroutine.

Input File Usage: **USER MATERIAL*

Abaqus/CAE Usage: Property module: material editor: **General**: **User Material**: **User material type: Thermal**

Forced Convection through the Mesh

The velocity of a fluid moving through the mesh can be prescribed if forced convection/diffusion heat transfer elements are used. Conduction between the fluid and adjacent forced convection/diffusion heat transfer elements will be affected by the mass flow rate of the fluid. For example, if a pipe is filled with a fluid with an initial temperature profile that contains a temperature pulse, the initial temperature pulse will not only diffuse (because of conduction in the fluid and the pipe), but it will also be transported (or convected) down the pipe. Since the fluid velocity is prescribed, it is called forced convection.

Natural convection occurs when differences in fluid density created by thermal gradients cause motion of the fluid (bouyancy-driven flow). The forced convection/diffusion elements are not designed to handle this phenomenon; the flow must be prescribed.

You can specify the mass flow rates per unit area (or through the entire section for one-dimensional elements) at the nodes. Abaqus/Standard interpolates the mass flow rates to the material points. The numerical solution of the transient heat transfer equation including convection becomes increasingly difficult as convection dominates diffusion. The Peclet number, γ, is a dimensionless parameter that indicates the degree of convection dominance over diffusion:

$$\gamma = |\mathbf{v}| \Delta \ell \frac{\rho c}{k},$$

where $|\mathbf{v}|$ is the magnitude of the velocity vector, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a characteristic element length in the direction of flow. Large values of γ indicate that convection dominates over diffusion on the spatial scale defined by the element size, $\Delta \ell$. In general, Peclet numbers greater than about 1000 should not be used.

Petrov-Galerkin finite elements are used in Abaqus/Standard to model systems with high Peclet numbers accurately; these elements use nonsymmetric, upwinded weighting functions to control numerical diffusion and dispersion and, thus, stabilize results. The upwinding term is partly a function of the element Peclet number, as described in *Convection/diffusion*.

If the fluid flows along a boundary along which a rapid change of temperature is prescribed, it is, in fact, subjected to a thermal transient, even for steady-state analysis. This transient can give rise to the same kind of spurious temperature oscillations that are observed in transient heat transfer analysis, as discussed earlier in this section. Since Abaqus/Standard uses first-order elements for convective heat transfer, the oscillation can be eliminated by lumping the heat capacity terms. However, the upwinded weighting functions prevent lumping in the direction of the flow. Hence, spurious oscillations may still occur, in particular if the flow is not precisely tangential to the boundary along which the temperature change occurs.

Input File Usage: Use the following option within the heat transfer step definition to prescribe the fluid velocity:

**MASS FLOW RATE*

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Modifying or Removing Mass Flow Rates

By default, the mass flow rates given are modifications of existing flow rates or are to be applied in addition to any mass flow rates defined previously. You can remove all previously defined mass flow rates and, optionally, specify new mass flow rates.

Input File Usage: Use the following option to modify an existing flow rate or to specify an additional flow rate:

**MASS FLOW RATE*, OP=MOD (default)

Use the following option to release all previously applied flow rates and to specify new flow rates:

**MASS FLOW RATE*, OP=NEW

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Specifying Time-Dependent Mass Flow Rates

Mass flow rates can be given in combination with an amplitude definition, if required, to control the magnitude of the flow rate as a function of time (*Amplitude Curves*).

Input File Usage: Use both of the following options to define a time-dependent mass flow rate:

**AMPLITUDE*, NAME=*name*
**MASS FLOW RATE*, AMPLITUDE=*name*

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Defining Mass Flow Rates in a User Subroutine

Mass flow rates can be defined by user subroutine *UMASFL*. *UMASFL* will be called for each specified node. Any mass flow rate values given directly will be ignored.

Input File Usage: **MASS FLOW RATE*, USER

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Reading the Mass Flow Rate Data from an Alternate File

The data for the mass flow rate can be contained in an alternate file. See *Input Syntax Rules* for the syntax of the file name.

Input File Usage: **MASS FLOW RATE*, INPUT=*file_name*

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Cavity Radiation

Cavity radiation can be activated in a heat transfer step. This feature involves interacting heat transfer between all of the facets of the cavity surface, dependent on the facet temperatures, facet emissivities, and the geometric view factors between each facet pair. When the thermal emissivity is a function of temperature or field variables, you can specify the maximum allowable emissivity change during an increment in addition to the maximum temperature change to control the time incrementation. See *Cavity Radiation in Abaqus/Standard* for more information.

Input File Usage: Use the following option in the step definition to activate cavity radiation:

**RADIATION VIEW FACTOR*

Use the following option to specify the maximum allowable emissivity change:

**HEAT TRANSFER*, MXDEM=*max_delta_emissivity*

Abaqus/CAE Usage: You can specify the maximum allowable emissivity change for a heat transfer step.

Step module: **Create Step**: **General**: **Heat transfer**: **Incrementation**: **Max. allowable emissivity change per increment**

Initial Conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures (see *Initial Conditions*).

Forced Convection through the Mesh

In an Abaqus/Standard heat transfer analysis involving forced convection through the mesh, you can define nonzero initial mass flow rates at the nodes of the forced convection/diffusion heat transfer elements in the model, as described in *Initial Conditions*.

For element types DCC1D2 and DCC1D2D the mass flow rate is positive from the first to the second node of the element. For two- and three-dimensional elements the direction of the mass flow rate is defined by giving the components in the *x*-, *y*-, and *z*-directions.

Input File Usage: **INITIAL CONDITIONS*, TYPE=MASS FLOW RATE

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Boundary Conditions

In Abaqus/Standard, boundary conditions can be used to prescribe temperatures (degree of freedom 11) at nodes in a heat transfer analysis (see *Boundary Conditions*). Shell elements have additional temperature degrees of freedom 12, 13, etc. through the thickness (see *Conventions*). Boundary conditions can be specified as functions of time by referring to amplitude curves (see *Amplitude Curves*).

For purely diffusive heat transfer elements a boundary without any prescribed boundary conditions (natural boundary condition) corresponds to an insulated surface. For forced convection/diffusion elements only the flux associated with conduction is zero; energy is free to convect across an unconstrained surface. This natural boundary condition correctly models areas where fluid is crossing a surface (as, for example, at the upstream and downstream boundaries of the mesh) and prevents spurious reflections of energy back into the mesh.

Loads

The following types of loading can be prescribed in a heat transfer analysis, as described in *Thermal Loads*:

- Concentrated heat fluxes (Abaqus/Standard only).
- Body fluxes and distributed surface fluxes.
- Average-temperature radiation conditions.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Cavity radiation effects can also be included in Abaqus/Standard, as described in *Cavity Radiation in Abaqus/Standard*.

Predefined Fields

Predefined temperature fields are not allowed in heat transfer analyses. Boundary conditions should be used instead to specify temperatures, as described earlier.

Other predefined field variables can be specified in a heat transfer analysis. These values will affect field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

The thermal conductivity of the materials in a heat transfer analysis must be defined. The specific heat and density of the materials must also be defined for transient heat transfer problems. Latent heat can be defined for diffusive heat transfer elements in Abaqus/Standard if changes in internal energy due to phase changes are

important. Latent heat cannot be defined directly for forced convection/diffusion elements. See *About Thermal Properties* for details on defining thermal properties in Abaqus.

Alternatively, user subroutine *UMATHT* can be used in Abaqus/Standard to define the thermal constitutive behavior of the material, including internal heat generation. For example, if a material modeled can go through a complex phase change, the specific heat can be defined in user subroutine *UMATHT* in sufficient detail to capture the phase change.

Thermal expansion coefficients are not meaningful in an uncoupled heat transfer analysis problem since deformation of the structure is not considered.

Elements

The heat transfer element library in Abaqus/Standard includes diffusive heat transfer elements, which allow for heat storage (specific heat and latent heat effects) and heat conduction.

Forced convection/diffusion heat transfer elements are also available in Abaqus/Standard: in addition to heat storage and heat conduction these elements allow for forced convection caused by fluid flowing through the mesh. These elements cannot be used with latent heat—see *Solid (Continuum) Elements* for additional details. Forced convection/diffusion elements with dispersion control are available for problems where the temperature transient in the fluid must be calculated accurately. See *Choosing the Appropriate Element for an Analysis Type*.

Multiple temperatures are available through the thickness of shell heat transfer elements in Abaqus/Standard. See *Choosing a Shell Element*.

The first-order heat transfer elements in Abaqus/Standard (such as the 2-node link, 4-node quadrilateral, and 8-node brick) use a numerical integration rule with the integration stations located at the corners of the element for the heat capacitance terms and for the calculations of the distributed body and surface fluxes. First-order diffusive elements are preferred in cases involving latent heat effects since they use such a special integration technique to provide accurate solutions with large latent heats. The forced convection/diffusion elements cannot use this special integration technique and, therefore, are unsuitable for problems with latent heat effects. The second-order heat transfer elements use conventional Gaussian integration. Thus, the second-order elements are to be preferred for problems when the solution will be smooth (without latent heat effects), and usually give more accurate results for the same number of nodes in the mesh.

Thermal interactions between adjacent surfaces and thermal gap elements are also provided in Abaqus/Standard to model heat transfer across the boundary layer between a solid and a fluid or between two closely adjacent solids. See *Thermal Contact Properties*.

Output

The following heat transfer output variables are available:

Element integration point variables:

HFL
Magnitude and components of the heat flux vector.

HFL*n*
Component n of the heat flux vector (n=1, 2, 3).

HFLM
Magnitude of the heat flux vector.

TEMP
Integration point temperatures. When heat capacity and conductivity contributions are integrated at different locations (for example, a mixed integration scheme which involves nodal integration for heat capacity versus Gauss-point integration for

conducivity contributions), the temperature output is always associated with the integration points where the heat capacity is integrated.

FV Integration point values of predefined field variables. The output of predefined field variables is always at the Gauss points. In particular if output at nodal locations is requested, the output would correspond to the predefined field variable values at the nearest Gauss points.

TEMPR Integration point temperature rate.

MFR User-specified mass flow rates.

MFR*n* Component *n* of the mass flow rate (*n*=1, 2, 3).

GRADT Temperature gradient vector.

GRADT*n* Component *n* of the temperature gradient vector (*n*=1,2,3).

Whole element variables:

FLUXS Current values of uniform distributed heat fluxes.

NFLUX Fluxes at the nodes caused by heat conduction (internal fluxes).

FILM Current values of film conditions.

RAD Current values of radiation conditions.

Nodal variables:

NT Nodal point temperatures.

NT*n* Temperature degree of freedom *n* at a node (*n*=11, 12, …).

RFL Reaction flux values due to prescribed temperature.

RFL*n* Reaction flux value *n* at a node (*n*=11, 12, …).

CFL Concentrated flux values.

CFL*n* Concentrated flux value *n* at a node (*n*=11, 12, …).

RFLE Total flux at a node, including flux convected through the node in forced convection/diffusion elements but excluding external fluxes due to user-defined concentrated fluxes, distributed fluxes, film conditions, radiation conditions, and cavity radiation. Since RFLE is a scalar nodal output variable, care should be taken when summing it over on two surfaces with shared nodes. If node sets on both surfaces include the shared nodes, the output of RFLE on the common nodes will contribute to the sums of this output quantity on both surfaces.

RFLE*n* Total flux value *n* at a node (*n*=11, 12, …).

All of the output variables available in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
…
*PHYSICAL CONSTANTS, ABSOLUTE ZERO=θ^Z
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data lines to prescribe initial temperatures at the nodes
*AMPLITUDE, NAME=trefamp
Data lines to define amplitude curve to be used for radiation reference temperature, θ^0
*FILM PROPERTY, NAME=film
Data lines to define the convection film coefficient, h, as a function of temperature
**
*STEP
Transient analysis including forced convection through the mesh
*HEAT TRANSFER, END=SS, DELTMX=Δθ_max
Data line to define incrementation and steady state
**
*CFLUX and/or *DFLUX
Data lines to define concentrated and/or distributed fluxes
*FILM
Data lines referring to film property table film
*RADIATE, AMPLITUDE=trefamp
Data lines to define boundary radiation
**
*EL PRINT
 TEMP, HFL, GRADT
 NFLUX, FILM, RAD
*NODE PRINT
 NT11, RFL
*END STEP
```

Fully Coupled Thermal-Stress Analysis

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

Overview

A fully coupled thermal-stress analysis:

- is performed when the mechanical and thermal solutions affect each other strongly and, therefore, must be obtained simultaneously;
- requires the existence of elements with both temperature and displacement degrees of freedom in the model;
- can be used to analyze time-dependent material response;
- cannot include cavity radiation effects but might include average-temperature radiation conditions (see *Thermal Loads*); and
- takes into account temperature dependence of material properties only for the properties that are assigned to elements with temperature degrees of freedom.

In Abaqus/Standard a fully coupled thermal-stress analysis:

- neglects inertia effects; and
- can be transient or steady-state.

In Abaqus/Explicit a fully coupled thermal-stress analysis:

- includes inertia effects; and
- models transient thermal response.

References:

- *Defining an Analysis*
- *About Heat Transfer Analysis Procedures*
- **COUPLED TEMPERATURE-DISPLACEMENT*
- **DYNAMIC TEMPERATURE-DISPLACEMENT*
- *Specifying an inelastic heat fraction*
- *Configuring a fully coupled, simultaneous heat transfer and stress procedure*
- *Configuring a dynamic fully coupled thermal-stress procedure using explicit integration*

Fully Coupled Thermal-Stress Analysis

Fully coupled thermal-stress analysis is needed when the stress analysis is dependent on the temperature distribution and the temperature distribution depends on the stress solution. For example, metalworking problems might include significant heating due to inelastic deformation of the material that, in turn, changes the material properties. In addition, contact conditions exist in some problems where the heat conducted between surfaces might depend strongly on the separation of the surfaces or the pressure transmitted across the surfaces (see *Thermal Contact Properties*). For such cases the thermal and mechanical solutions must be obtained simultaneously rather than sequentially. Coupled temperature-displacement elements are provided for this purpose in both Abaqus/Standard and Abaqus/Explicit; however, each program uses different algorithms to solve coupled thermal-stress problems.

Fully Coupled Thermal-Stress Analysis in Abaqus/Standard

In Abaqus/Standard the temperatures are integrated using a backward-difference scheme, and the nonlinear coupled system is solved using Newton's method. Abaqus/Standard offers an exact as well as an approximate implementation of Newton's method for fully coupled temperature-displacement analysis.

Exact Implementation

An exact implementation of Newton's method involves a nonsymmetric Jacobian matrix as is illustrated in the following matrix representation of the coupled equations:

$$\begin{bmatrix} K_{uu} & K_{u\theta} \\ K_{\theta u} & K_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta \theta \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_\theta \end{Bmatrix},$$

where Δu and $\Delta \theta$ are the respective corrections to the incremental displacement and temperature, K_{ij} are submatrices of the fully coupled Jacobian matrix, and R_u and R_θ are the mechanical and thermal residual vectors, respectively.

Solving this system of equations requires the use of the unsymmetric matrix storage and solution scheme. In addition, the mechanical and thermal equations must be solved simultaneously. The method provides quadratic convergence when the solution estimate is within the radius of convergence of the algorithm. The exact implementation is used by default.

Approximate Implementation

Some problems require a fully coupled analysis in the sense that the mechanical and thermal solutions evolve simultaneously, but with a weak coupling between the two solutions. In other words, the components in the off-diagonal submatrices $K_{u\theta}$, $K_{\theta u}$ are small compared to the components in the diagonal submatrices K_{uu} , $K_{\theta\theta}$. An example of such a situation is the disc brake problem (*Thermal-stress analysis of a disc brake*). For these problems a less costly solution might be obtained by setting the off-diagonal submatrices to zero so that we obtain an approximate set of equations:

$$\begin{bmatrix} K_{uu} & 0 \\ 0 & K_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta \theta \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_\theta \end{Bmatrix}.$$

As a result of this approximation the thermal and mechanical equations can be solved separately, with fewer equations to consider in each subproblem. The savings due to this approximation, measured as solver time per iteration, will be of the order of a factor of two, with similar significant savings in solver storage of the factored stiffness matrix. Further, in many situations the subproblems might be fully symmetric or approximated as symmetric, so that the less costly symmetric storage and solution scheme can be used. The solver time savings for a symmetric solution is an additional factor of two. Unless you explicitly choose the unsymmetric matrix storage and solution scheme, selection of the scheme depends on other details of the problem (see *Defining an Analysis*).

This modified form of Newton's method does not affect solution accuracy since the fully coupled effect is considered through the residual vector R_j at each increment in time. However, the rate of convergence is no longer quadratic and depends strongly on the magnitude of the coupling effect, so more iterations are generally needed to achieve equilibrium than with the exact implementation of Newton's method. When the coupling is significant, the convergence rate becomes very slow and might prohibit obtaining a solution. In such cases the exact implementation of Newton's method is required. In cases where it is possible to use this approximation, the convergence in an increment depends strongly on the quality of the first guess to the incremental solution, which you can control by selecting the extrapolation method used for the step (see *Defining an Analysis*).

Input File Usage: Use the following option to specify a separated solution scheme:

*SOLUTION TECHNIQUE, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Other**: **Solution technique: Separated**

Steady-State Analysis

A steady-state coupled temperature-displacement analysis can be performed in Abaqus/Standard. In steady-state cases you should assign an arbitrary "time" scale to the step: you specify a "time" period and "time" incrementation parameters. This time scale is convenient for changing loads and boundary conditions through the step and for obtaining solutions to highly nonlinear (but steady-state) cases; however, for the latter purpose, transient analysis often provides a natural way of coping with the nonlinearity.

Frictional slip heat generation is normally neglected in for the steady-state case. However, it can still be accounted for if user subroutine *FRIC* provides the incremental frictional dissipation through the variable `SFD`. If frictional heat generation is present, the heat flux into the two contact surfaces depends on the slip rate of the surfaces. The "time" scale in this case cannot be described as arbitrary, and a transient analysis should be performed.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Steady state**

Transient Analysis

Alternatively, you can perform a transient coupled temperature-displacement analysis. You can control the time incrementation in a transient analysis directly, or Abaqus/Standard can control it automatically. Automatic time incrementation is generally preferred.

Automatic Incrementation Controlled by a Maximum Allowable Temperature Change

The time increments can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see *Time Integration Accuracy in Transient Problems*).

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**, **Max. allowable temperature change per increment: $\Delta\theta_{max}$**

Fixed Incrementation

If you do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, is used throughout the analysis, except when the explicit creep integration scheme is used. In this case Abaqus/Standard might decrease the time increment if the stability limit is exceeded.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious Oscillations due to Small Time Increments

In transient analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta \ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and might cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Automatic Incrementation Controlled by the Creep Response

The accuracy of the integration of time-dependent (creep) material behavior is governed by the user-specified accuracy tolerance parameter, $tolerance \geq \left(\dot{\bar{\varepsilon}}^{cr}{}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}{}_{t}\right)\Delta t$. This parameter is used to prescribe the maximum strain rate change allowed at any point during an increment, as described in *Rate-Dependent Plasticity: Creep and Swelling*. The accuracy tolerance parameter can be specified together with the maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$ (described above); however, specifying the accuracy tolerance parameter activates automatic incrementation even if $\Delta\theta_{max}$ is not specified.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, DELTMX=$\Delta\theta_{max}$, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**, **Max. allowable temperature change per increment:** $\Delta\theta_{max}$, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Selecting Explicit Creep Integration

Nonlinear creep problems (*Rate-Dependent Plasticity: Creep and Swelling*) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required if no other nonlinearities are present. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For most coupled thermal-stress analyses, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases the implicit integration scheme might be invoked automatically by Abaqus/Standard.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine *CREEP*; you can restrict Abaqus/Standard to using this method for creep problems (with or without geometric nonlinearity included). See *Rate-Dependent Plasticity: Creep and Swelling* for further details.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*, **Creep/swelling/viscoelastic integration: Explicit**

Excluding Creep and Viscoelastic Response

You can specify that no creep or viscoelastic response occurs during a step even if creep or viscoelastic material properties have been defined.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, DELTMX=$\Delta\theta_{max}$, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable Problems

Some types of analyses might develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it might not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers a method of stabilizing this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Units

In coupled problems where two different fields are active, take care when choosing the units of the problem. If the choice of units is such that the terms generated by the equations for each field are different by many orders of magnitude, the precision on some computers might be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid ill-conditioned matrices. For example, consider using units of megapascal (MPa) instead of pascal (Pa) for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations and the heat flux continuity equations.

Fully Coupled Thermal-Stress Analysis in Abaqus/Explicit

In Abaqus/Explicit the heat transfer equations are integrated using the explicit forward-difference time integration rule

$$\theta^N_{(i+1)} = \theta^N_{(i)} + \Delta t_{(i+1)} \dot{\theta}^N_{(i)},$$

where θ^N is the temperature at node N and the subscript i refers to the increment number in an explicit dynamic step. The forward-difference integration is explicit in the sense that no equations need to be solved when a lumped capacitance matrix is used. The current temperatures are obtained using known values of $\dot{\theta}^N_{(i)}$ from the previous increment. The values of $\dot{\theta}^N_{(i)}$ are computed at the beginning of the increment by

$$\dot{\theta}^N_{(i)} = \left(C^{NJ}\right)^{-1} \left(P^J_{(i)} - F^J_{(i)}\right),$$

where C^{NJ} is the lumped capacitance matrix, P^J is the applied nodal source vector, and F^J is the internal flux vector.

The mechanical solution response is obtained using the explicit central-difference integration rule with a lumped mass matrix as described in *Explicit Dynamic Analysis*. Since both the forward-difference and central-difference integrations are explicit, the heat transfer and mechanical solutions are obtained simultaneously by an explicit coupling. Therefore, no iterations or tangent stiffness matrices are required.

Explicit integration can be less expensive computationally and simplifies the treatment of contact. For a comparison of explicit and implicit direct-integration procedures, see *About Dynamic Analysis Procedures*.

Stability

The explicit procedure integrates through time by using many small time increments. The central-difference and forward-difference operators are conditionally stable. The stability limit for both operators (with no damping in the mechanical solution response) is obtained by choosing

$$\Delta t \leq min\left(\frac{2}{\omega_{max}}, \frac{2}{\lambda_{max}}\right),$$

where ω_{max} is the highest frequency in the system of equations of the mechanical solution response and λ_{max} is the largest eigenvalue in the system of equations of the thermal solution response.

Estimating the Time Increment Size

An approximation to the stability limit for the forward-difference operator in the thermal solution response is given by

$$\Delta t \approx \frac{L^2_{min}}{2\alpha},$$

where L_{min} is the smallest element dimension in the mesh and $\alpha = \frac{k}{\rho c}$ is the thermal diffusivity of the material. The parameters k, ρ, and c represent the material's thermal conductivity, density, and specific heat, respectively.

In most applications of explicit analysis the mechanical response governs the stability limit. The thermal response might govern the stability limit when material parameter values are nonphysical or a very large amount of mass scaling is used. The calculation of the time increment size for the mechanical solution response is discussed in *Explicit Dynamic Analysis*.

Stable Time Increment Report

Abaqus/Explicit writes a report to the status (`.sta`) file during the data check phase of the analysis that contains an estimate of the minimum stable time increment and a listing of the elements with the smallest stable time increments and their values. The initial minimum stable time increment accounts for the stability requirements of both the thermal and mechanical solution responses. The initial stable time increments listed do not include damping (bulk viscosity), mass scaling, or penalty contact effects in the mechanical solution response.

This listing is provided because often a few elements have much smaller stability limits than the rest of the elements in the mesh. The stable time increment can be increased by modifying the mesh to increase the size of the controlling element or by using appropriate mass scaling.

Time Incrementation

The time increment used in an analysis must be smaller than the stability limits of the central- and forward-difference operators. Failure to use such a time increment results in an unstable solution. When the solution becomes unstable, the time history response of solution variables, such as displacements, usually oscillates with increasing amplitudes. The total energy balance also changes significantly.

Abaqus/Explicit has two strategies for time incrementation control: fully automatic time incrementation (where the code accounts for changes in the stability limit) and fixed time incrementation.

Scaling the Time Increment

To reduce the chance of a solution going unstable, the stable time increment computed by Abaqus/Explicit can be adjusted by a constant scaling factor. This factor can be used to scale the default global time estimate, the element-by-element estimate, or the fixed time increment based on the initial element-by-element estimate; it cannot be used to scale a fixed time increment that you specified directly.

Input File Usage: Use any of the following options:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, SCALE FACTOR=*f*
**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT,
ELEMENT BY ELEMENT, SCALE FACTOR=*f*
**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT,
FIXED TIME INCREMENTATION, SCALE FACTOR=*f*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Time scaling factor:** *f*

Automatic Time Incrementation

The default time incrementation scheme in Abaqus/Explicit is fully automatic and requires no user intervention. Two types of estimates are used to determine the stability limit: element-by-element for both the thermal and mechanical solution responses and global for the mechanical solution response. An analysis always starts by using the element-by-element estimation method and might switch to the global estimation method under certain circumstances, as explained in *Explicit Dynamic Analysis*.

In an analysis Abaqus/Explicit initially uses a stability limit based on the thermal and mechanical solution responses in the whole model. This element-by-element estimate is determined using the smallest time increment size due to the thermal and mechanical solution responses in each element.

The element-by-element estimate is conservative; it gives a smaller stable time increment than the true stability limit, which is based on the maximum frequency of the entire model. In general, constraints such as boundary conditions and kinematic contact have the effect of compressing the eigenvalue spectrum, and the element-by-element estimates do not take this into account (see *Explicit Dynamic Analysis*)

The stable time increment size due to the mechanical solution response is determined by the global estimator as the step proceeds unless the element-by-element estimator is chosen, fixed time incrementation is specified, or one of the conditions explained in *Explicit Dynamic Analysis* prevents the use of global estimation. The stable time increment size due to the thermal solution response is always determined by using an element-by-element estimation method. The switch to the global estimation method in mechanical solution response occurs once the

algorithm determines that the accuracy of the global estimation method is acceptable. For details, see *Explicit Dynamic Analysis*

For three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements) an "improved" estimate of the element characteristic length is used by default. This "improved" method usually results in a larger element stable time increment than a more traditional method. For analyses using variable mass scaling, the total mass added to achieve a given stable time increment is less with the improved estimate.

Input File Usage: Use the following option to specify the element-by-element estimation method:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, ELEMENT BY ELEMENT

Use the following option to activate the "improved" element time estimation method:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, IMPROVED DT METHOD=YES

Use the following option to deactivate the "improved" element time estimation method:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, IMPROVED DT METHOD=NO

Abaqus/CAE Usage: Use the following option to specify the element-by-element estimation method:

Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Automatic**, **Stable increment estimator: Element-by-element**

Use the following option to activate the "improved" element time estimation method:

Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Automatic**: toggle on **Improved Dt Method**

Use the following option to deactivate the "improved" element time estimation method:

Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Automatic**: toggle off **Improved Dt Method**

Fixed Time Incrementation

A fixed time incrementation scheme is also available in Abaqus/Explicit. The fixed time increment size is determined either by the initial element-by-element stability estimate for the step or by a user-specified time increment.

Fixed time incrementation might be useful when a more accurate representation of the higher mode response of a problem is required. In this case a time increment size smaller than the element-by-element estimates might be used. The element-by-element estimate can be obtained by running a data check analysis (see *Abaqus/Standard and Abaqus/Explicit Execution*).

When fixed time incrementation is used, Abaqus/Explicit does not check that the computed response is stable during the step. You should ensure that a valid response has been obtained by carefully checking the energy history and other response variables.

If you choose to use time increments the size of the initial element-by-element stability limit throughout a step, the dilatational wave speed and the thermal diffusivity in each element at the beginning of the step are used to compute the fixed time increment size. To reduce the chance of a solution going unstable, the initial stable time increment that Abaqus/Explicit computes can be adjusted by a constant scaling factor, as described above in *Scaling the Time Increment*. Alternatively, you can specify a time increment size directly.

Input File Usage: Use the following option to request time increments the size of the element-by-element stability limit:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT,
FIXED TIME INCREMENTATION

Use the following option to specify the time increment size directly:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, DIRECT USER CONTROL
Δt

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Fixed**, **Use element-by-element time increment estimator** or **User-defined time increment**: Δt

Reducing the Computational Cost by Using Selective Subcycling

The selective subcycling method can be used in a coupled thermal-stress analysis exactly as in a pure mechanical analysis, as described in *Explicit Dynamic Analysis* and *Selective Subcycling*.

Monitoring Output Variables for Extreme Values

The extreme values defined as the element and nodal variables in a coupled thermal-stress analysis can be monitored exactly as described in *Explicit Dynamic Analysis* for a pure mechanical analysis.

Controlling Material and Friction Rate Dependence

You can control material and friction rate dependence.

Rate-Dependent Yield and Friction in Abaqus/Standard

In Abaqus/Standard, you can control whether to consider or ignore the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient within the step.

Input File Usage: Use the following option to consider rate dependence within the step (default):

**COUPLED TEMPERATURE-DISPLACEMENT*, RATE DEPENDENCE=ON

Use the following option to ignore rate dependence within the step:

**COUPLED TEMPERATURE-DISPLACEMENT*, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient.

Material Rate Dependence in Abaqus/Explicit

In Abaqus/Explicit, you can control whether to consider or ignore material rate dependence within the step. A step that ignores material rate dependence should not follow a step or analysis that considers material rate dependence.

The fully relaxed long-term elastic response is used for viscoelastic materials when the step ignores material rate dependence. The rate dependence of the material response defined via user subroutines *VUCREEPNETWORK*, *VUHARD*, and *VUMAT* can be ignored when the step ignores material rate dependence.

Input File Usage: Use the following option to consider material rate dependence within the step (default):

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, RATE DEPENDENCE=ON

Use the following option to ignore material rate dependence within the step:

**DYNAMIC TEMPERATURE-DISPLACEMENT*, EXPLICIT, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers material rate dependence.

Initial Conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures. Initial stresses, field variables, etc. can also be defined; *Initial Conditions* describes all the initial conditions that are available for a fully coupled thermal-stress analysis.

Boundary Conditions

Boundary conditions can be used to prescribe both temperatures (degree of freedom 11) and displacements/rotations (degrees of freedom 1–6) at nodes in fully coupled thermal-stress analysis (see *Boundary Conditions*). Shell elements in Abaqus/Standard have additional temperature degrees of freedom 12, 13, etc. through the thickness (see *Conventions*).

Boundary conditions can be specified as functions of time by referring to amplitude curves (*Amplitude Curves*).

Boundary conditions applied during a dynamic coupled temperature-displacement response step should use appropriate amplitude references (*Amplitude Curves*). If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference is ignored, and a zero velocity boundary condition is enforced.

Loads

The following types of thermal loads can be prescribed in a fully coupled thermal-stress analysis, as described in *Thermal Loads*:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Node-based film and radiation conditions.
- Average-temperature radiation conditions.

- Element and surface-based film and radiation conditions.

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Predefined Fields

Predefined temperature fields are not allowed in a fully coupled thermal-stress analysis. Boundary conditions should be used instead to prescribe temperature degree of freedom 11 (and 12, 13, etc. in Abaqus/Standard shell elements), as described earlier.

Other predefined field variables can be specified in a fully coupled thermal-stress analysis. These values affect only field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

The materials in a fully coupled thermal-stress analysis must have both thermal properties, such as conductivity, and mechanical properties, such as elasticity, defined. See *Abaqus Materials Guide* for details on the material models available in Abaqus.

Thermal strain arises if thermal expansion (*Thermal Expansion*) is included in the material property definition.

In Abaqus/Standard a fully coupled temperature-displacement analysis can be used to analyze static creep and swelling problems, which generally occur over fairly long time periods (*Rate-Dependent Plasticity: Creep and Swelling*); viscoelastic materials (*Time Domain Viscoelasticity*); or viscoplastic materials (*Rate-Dependent Yield*).

Internal Heat Generation

In Abaqus/Standard analyses, you can define volumetric heat generation within a material in user subroutine *HETVAL* or user subroutine *UMATHT*. You can use these two user subroutines in the same analysis.

In Abaqus/Explicit analyses, you can define volumetric heat generation within a material in user subroutine *VHETVAL* or user subroutine *VUMATHT*. You can use these two user subroutines in the same analysis.

Defining Internal Heat Generation in User Subroutine `HETVAL` or `VHETVAL`

If you define internal heat generation in user subroutine *HETVAL* or *VHETVAL*, you must include heat generation in the material definition with the other thermal property definitions.

Heat generation might be associated with (relatively low) energy phase changes occurring during the solution. Such heat generation usually depends on state variables (such as the fraction transformed), which themselves evolve with the solution and are stored as solution-dependent state variables (see *About User Subroutines and Utilities*). The heat generation is computed in user subroutine *HETVAL* or *VHETVAL*, where any associated state variables can also be updated. The subroutine is called at all material calculation points for which the material definition includes heat generation.

Input File Usage: **HEAT GENERATION*

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Heat Generation**

Defining Internal Heat Generation in User Subroutine UMATHT or VUMATHT

If user subroutine *UMATHT* or *VUMATHT* is used to define internal heat generation, the constitutive thermal behavior must also be defined within the subroutine.

Input File Usage: **USER MATERIAL*

Abaqus/CAE Usage: Property module: material editor: **General**: **User Material**: **User material type: Thermal**

Inelastic Energy Dissipation as a Heat Source

You can specify an inelastic heat fraction in a fully coupled thermal-stress analysis to provide for inelastic energy dissipation as a heat source. The heat flux per unit volume, r^{pl}, that is added into the thermal energy balance is computed using the equation

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

or, in the case when the nonlinear isotropic/kinematic hardening model is used, from the following equation:

$$r^{pl} = \eta (\boldsymbol{\sigma} - \boldsymbol{\alpha}) : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where η is a user-defined factor (assumed constant), $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\alpha}$ is the backstress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining.

Inelastic heat fractions are typically used in the simulation of high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation significantly influences temperature-dependent material properties. The generated heat is treated as a volumetric heat flux source term in the heat balance equation.

An inelastic heat fraction can be specified for materials with plastic behavior that use the Mises or Hill yield surface (*Inelastic Behavior*), and it can also be used with the combined isotropic/kinematic hardening model. The inelastic heat fraction can be specified for user-defined material behavior in Abaqus/Explicit and is multiplied by the inelastic energy dissipation coded in the user subroutine to obtain the heat flux. In Abaqus/Standard the inelastic heat fraction cannot be used with user-defined material behavior; in this case the heat flux that must be added to the thermal energy balance is computed directly in the user subroutine.

An inelastic heat fraction can also be specified for material definitions that include time-domain linear viscoelasticity (*Time Domain Viscoelasticity*) and time-domain nonlinear viscoelasticity defined within the parallel rheological framework (*Parallel Rheological Framework*), except in Abaqus/Explicit for large-strain linear viscoelasticity. For large-strain linear viscoelasticity in Abaqus/Standard (*Time Domain Viscoelasticity*), the energy dissipation is computed only approximately. Hence, the fraction of the dissipated energy converted into heat can be computed only approximately.

The default value of the inelastic heat fraction is 0.9. If you do not include the inelastic heat fraction behavior in the material definition, the heat generated by inelastic deformation is not included in the analysis.

Input File Usage: **INELASTIC HEAT FRACTION*
η

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Inelastic Heat Fraction**: **Fraction:** η

Elements

Coupled temperature-displacement elements that have both displacements and temperatures as nodal variables are available in both Abaqus/Standard and Abaqus/Explicit (see *Choosing the Appropriate Element for an Analysis Type*). In Abaqus/Standard simultaneous temperature/displacement solution requires the use of such elements; pure displacement elements can be used in part of the model in the fully coupled thermal-stress procedure, but pure heat transfer elements cannot be used. In Abaqus/Explicit any of the available elements can be used in the fully coupled thermal-stress procedure; however, the thermal solution is obtained only at nodes where the temperature degree of freedom has been activated (that is, at nodes attached to coupled temperature-displacement elements).

The first-order coupled temperature-displacement elements in Abaqus use a constant temperature over the element to calculate thermal expansion. The second-order coupled temperature-displacement elements in Abaqus/Standard use a lower-order interpolation for temperature than for displacement (parabolic variation of displacements and linear variation of temperature) to obtain a compatible variation of thermal and mechanical strain.

Output

See *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers* for a complete list of output variables. The types of output available are described in *About Output*.

Input File Template

```
*HEADING
…
** Specify the coupled temperature-displacement element type
*ELEMENT, TYPE=CPS4T
…
**
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT or
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
Data line to define incrementation
*BOUNDARY
Data lines to define nonzero boundary conditions on displacement or
temperature degrees of freedom
*CFLUX and/or *CFILM and/or
*CRADIATE and/or *DFLUX and/or
*DSFLUX and/or *FILM and/or
*SFILM and/or *RADIATE and/or
*SRADIATE
Data lines to define thermal loads
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to define mechanical loads
*FIELD
Data lines to define field variable values
*END STEP
```

Adiabatic Analysis

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

Overview

An adiabatic stress analysis:

- is used in cases where mechanical deformation causes heating but the event is so rapid that this heat has no time to diffuse through the material—for example, a very high-speed forming process;
- can be conducted as part of a dynamic analysis (*Implicit Dynamic Analysis Using Direct Integration* or *Explicit Dynamic Analysis*) or as part of a static analysis (*Static Stress Analysis*);
- in Abaqus/Standard is available only for the isotropic hardening metal plasticity models with a Mises yield surface (*Classical Metal Plasticity*);
- in Abaqus/Explicit is relevant only for the metal plasticity models (including both Mises and Hill yield surfaces);
- can be conducted if parts of the model are elastic only—no change in temperature occurs in the elastic regions; and
- requires that a material's density, specific heat, and inelastic heat fraction (fraction of inelastic dissipation rate that appears as heat flux) be specified.

References:

- *Defining an Analysis*
- *About Heat Transfer Analysis Procedures*
- **DYNAMIC*
- **STATIC*
- **DENSITY*
- **INELASTIC HEAT FRACTION*
- **SPECIFIC HEAT*
- *Configuring general analysis procedures*
- *Defining thermal material models*

Adiabatic Analysis

Adiabatic thermal-stress analysis is typically used to simulate high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation is an important effect because of temperature-dependent material properties. The temperature increase is calculated directly at the material integration points according to the adiabatic thermal energy increases caused by inelastic deformation; temperature is not a degree of freedom in the problem. No allowance is made for conduction of heat in an adiabatic analysis. For problems where both inelastic heating and conduction of the heat are important, a fully coupled temperature-displacement analysis must be performed (*Fully Coupled Thermal-Stress Analysis*).

In an adiabatic analysis plastic straining gives rise to a heat flux per unit volume of

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where r^{pl} is the heat flux that is added into the thermal energy balance, η is the user-specified inelastic heat fraction (assumed constant; discussed below), $\boldsymbol{\sigma}$ is the stress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining. The heat equation solved at each integration point is

$$\rho c(\theta)\dot{\theta} = r^{pl},$$

where ρ is the material density and $c(\theta)$ is the specific heat (see *Density* and *Specific Heat*).

Input File Usage: Use any of the following procedures to perform an adiabatic analysis:

**DYNAMIC*, ADIABATIC
**DYNAMIC*, EXPLICIT, ADIABATIC
**STATIC*, ADIABATIC

Abaqus/CAE Usage: Use any of the following procedures to perform an adiabatic analysis:

Step module:
Create Step: **Dynamic, Implicit**: **Basic**: **Include adiabatic heating effects**
Create Step: **Dynamic, Explicit**: **Basic**: **Include adiabatic heating effects**
Create Step: **Static, General**: **Basic**: **Include adiabatic heating effects**

Subsequent Thermal Diffusion Analysis in Abaqus/Standard

In Abaqus/Standard thermal diffusion analysis can be performed after the adiabatic calculation (for example, to study the cool-down of a component after sudden deformation). In this case the temperatures at the end of the adiabatic analysis must be written to the Abaqus/Standard results file as element variables averaged at the nodes. Since temperature values in an adiabatic analysis can be written to the results file as element quantities only by using the TEMP output variable identifier, they cannot be read directly into a subsequent thermal diffusion analysis as initial conditions. However, if you postprocess the results file to produce a second results file in which the temperature data are provided as nodal quantities, a subsequent heat transfer analysis can be performed with these temperatures as initial conditions. See *Predefined Fields* and *Accessing the Results File Information* for details. Alternatively, you could postprocess the results file to produce a data list containing data pairs consisting of nodes and temperatures.

The temperatures, NT, obtained from the heat transfer analysis can then be used to drive a continuation of the previous stress analysis. This stress analysis should be restarted from the end of the adiabatic analysis and will provide the response to the change of the temperature field obtained during the heat transfer analysis. In this case Abaqus/Standard will automatically read the temperatures from the results file that was obtained from the heat transfer analysis and apply them in the restarted analysis.

Example

The following input options could be used to perform a heat transfer analysis using the temperatures from an adiabatic analysis and then continue the stress analysis:

```
**Static adiabatic analysis
 …
*STEP
*STATIC, ADIABATIC
 …
**Write the temperatures to the results file as element
**variables averaged at the nodes
*EL FILE, POSITION=AVERAGED AT NODES
 TEMP
*END STEP
```

```
**Heat transfer analysis using the temperatures from the
**static analysis as initial conditions
 …
*INITIAL CONDITIONS, TYPE=TEMPERATURE, FILE=new results file,
 STEP=step, INC=increment
*STEP
*HEAT TRANSFER
 …
*NODE FILE
 NT
*END STEP
**Restart from the adiabatic analysis using temperatures
**obtained from the heat transfer analysis
*RESTART, WRITE, READ, STEP=k, INC=i, END STEP
 …
*STEP
*STATIC
 …
*TEMPERATURE, FILE=heat_transfer_results_file
 …
*END STEP
```

Fully Coupled Temperature-Displacement Analysis

If the continuation of the analysis into thermal diffusion requires a fully coupled temperature-displacement analysis (see *Fully Coupled Thermal-Stress Analysis*), the simplest (but more expensive) approach is to use coupled temperature-displacement elements throughout the adiabatic analysis. At the end of the static or the dynamic adiabatic calculations, the temperatures must be written to the results file as element variables averaged at the nodes. In addition, you must constrain all temperature degrees of freedom since they are not used in the adiabatic analysis. The adiabatic analysis can then be restarted to apply the correct temperature distribution obtained from the adiabatic analysis to the temperature degree of freedom of each node in the model. To create the input for the boundary conditions, you must postprocess the results file obtained from the adiabatic analysis and extract the value of TEMP at each node in the model (see *Accessing the Results File Information*). The temperature boundary conditions can be released as needed in subsequent coupled temperature-displacement analysis steps.

Example

The following input options could be used to perform a coupled temperature-displacement analysis using the temperatures from an adiabatic analysis:

```
**Static adiabatic analysis, coupled temperature-displacement
**plane stress elements
 …
*ELEMENT, TYPE=CPS4T, ELSET=EALL
 …
*BOUNDARY
nodes, 11, 11, 0.0
*STEP
*STATIC, ADIABATIC
 …
**Write the temperatures to the results file as element
**variables averaged at the nodes
*EL FILE, POSITION=AVERAGED AT NODES
 TEMP
*END STEP
**Restart from the adiabatic analysis
*RESTART, WRITE, READ, STEP=k, INC=i, END STEP
```

```
 …
*STEP
*STATIC
**Dummy step to associate the temperature variable TEMP with
**the temperature degree of freedom at each node
 1.0, 1.0
 …
*BOUNDARY, OP=NEW
 node, 11, 11, temperature
 …
*END STEP
**Coupled temperature displacement run for cool down of
**structure: continuation of the restart analysis
 …
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT
 0.1, 1.0
 …
*BOUNDARY, OP=NEW
**no temperature boundary condition specified
*END STEP
```

Initial Conditions

Initial temperatures can be prescribed at nodes as initial conditions. Initial values of stresses, field variables, solution-dependent state variables, etc. can also be specified (see *Initial Conditions*).

Boundary Conditions

Boundary conditions can be applied to displacement degrees of freedom in an adiabatic analysis in the same way that they are applied in nonadiabatic dynamic, explicit dynamic, or static analysis steps (see *Boundary Conditions*). Temperature is not a degree of freedom in an adiabatic analysis.

Loads

The loading options available for an adiabatic analysis are the same as those available for nonadiabatic dynamic, explicit dynamic, or static analysis steps (see *About Loads*).

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

Predefined Fields

Predefined temperature fields cannot be used during an adiabatic analysis step.

The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

In Abaqus/Standard only Mises plasticity with isotropic elasticity and isotropic hardening (*Inelastic Behavior*) is allowed in adiabatic stress analysis. Kinematic or combined hardening is not available, but rate effects can be included. However, portions of the model can include only elastic material; no change in temperature occurs in the elastic regions, since there is no source of heat generation. In Abaqus/Explicit both Mises and Hill plasticity are allowed in adiabatic stress analysis.

You must specify the density, the inelastic heat fraction, and the specific heat as part of the material definition for the material in which heat will be generated by plastic dissipation. You can also specify latent heat if necessary (*Latent Heat*).

The inelastic heat fraction is the amount of inelastic dissipation used to calculate the increase in temperature. The default value of the inelastic heat fraction is 0.9. If the inelastic heat fraction is not included in the material definition, the heat generated by inelastic deformation is not included in the analysis.

In Abaqus/Standard adiabatic analyses can also be carried out with user subroutine *UMAT*. In this case the temperature must be defined as a solution-dependent state variable, and all coupling terms must be included in the user subroutine. If conductivity (*Conductivity*) is defined for the material, it will be ignored during adiabatic analysis steps.

Input File Usage: All of the following options must be included in the material definition:

**DENSITY*
**INELASTIC HEAT FRACTION*
**SPECIFIC HEAT*

The following option can be included if latent heat effects are important:

**LATENT HEAT*

Abaqus/CAE Usage: All of the following must be included in the material definition:

Property module:
Material editor: **General->Density**
Material editor: **Thermal->Inelastic Heat Fraction**
Material editor: **Thermal->Specific Heat**

The following can be included if latent heat effects are important:

Property module: material editor: **Thermal->Latent Heat**

Temperature-Dependent Material Properties

Material properties can be temperature dependent. Since the only source of temperature change in adiabatic analysis is inelastic deformation, the temperature can only rise. This temperature rise may cause thermal expansion (usually a small effect) and localization of the deformation if the flow stress is reduced by the temperature rise. Since the adiabatic assumption applies only in rapid events and inelastic deformation usually causes significant temperature rises only if the deformation is substantial, the strain rates are often large in adiabatic analysis. The softening of the material caused by the temperature rise may, thus, be offset somewhat by strengthening associated with rate dependence if the material is rate sensitive.

Elements

Any of the stress/displacement or coupled temperature-displacement elements in Abaqus can be used in an adiabatic analysis (see *Choosing the Appropriate Element for an Analysis Type*). Mass or spring elements will not contribute to the heating of the material since they cannot generate plastic strains.

If coupled temperature-displacement elements are used in an adiabatic analysis, the temperature degrees of freedom will be ignored.

Output

Since temperatures are updated at the material calculation points, output of temperature is available with output variable TEMP, not with output variable NT.

The element output available for an adiabatic analysis includes stress; strain; energies; the values of state, field, and user-defined variables; and composite failure measures. The nodal output available includes displacements, reaction forces, and coordinates. All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers*.

Input File Template

```
*HEADING
…
*MATERIAL, NAME=name
*ELASTIC, TYPE=ISOTROPIC
Data lines to define isotropic linear elasticity
*PLASTIC
Data lines to define metal plasticity
*DENSITY
Data lines to define density
*INELASTIC HEAT FRACTION
Data line to define inelastic heat fraction
*SPECIFIC HEAT
Data lines to define specific heat
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=type
Data lines to specify initial conditions
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**


*STEP, NLGEOM
The NLGEOM parameter is used in Abaqus/Standard to include geometric nonlinearity
*DYNAMIC, ADIABATIC or *DYNAMIC, EXPLICIT, ADIABATIC or
*STATIC, ADIABATIC
Data line to control time incrementation or to specify the time period of the step
*BOUNDARY, AMPLITUDE=name
Data lines to describe nonzero or zero-valued boundary conditions
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to specify loads
*FIELD
Data lines to specify field variable values
*END STEP
```

Electromagnetic Analysis

In this section:

- *Electromagnetic Analysis Procedures*
- *Piezoelectric Analysis*
- *Coupled Thermal-Electrical Analysis*
- *Fully Coupled Thermal-Electrical-Structural Analysis*
- *Eddy Current Analysis*
- *Magnetostatic Analysis*

Electromagnetic Analysis Procedures

Overview

Abaqus/Standard offers several analysis procedures to model piezoelectric, electrical conduction, and electromagnetic phenomena. The distinct electrical phenomena modeled by these procedures is described first, followed by a brief overview of each procedure.

References:

Electrostatic, Electrical Conduction, Magnetostatic, and Electromagnetic Analyses

Piezoelectric effect is the electromechanical interaction exhibited by some materials. This coupled electrostatic-structural response is modeled using piezoelectric analysis in Abaqus/Standard. In this procedure the electric potential is a degree of freedom and its conjugate is the electric charge.

Coupled thermal-electrical conduction, with or without structural coupling, is modeled using electrical procedures. In these procedures the electric potential is a degree of freedom and its conjugate is the electric current. While transient effects are ignored in electrical conduction, thus making it steady state, thermal fields can be modeled either as transient or steady state.

Magnetostatic analysis is used to compute the magnetic fields due to direct currents. It solves the magnetostatic approximation to Maxwell's equations. The magnetic vector potential is a degree of freedom in a magnetostatic analysis, and its conjugate is the surface current.

Electromagnetic analysis is used to model the full coupling between time-varying electric and magnetic fields by solving Maxwell's equations. In such an analysis the magnetic vector potential is a degree of freedom and its conjugate is the surface current.

Electrostatic Procedure

The following electrostatic analysis procedure is available in Abaqus/Standard:

Piezoelectric analysis

In a piezoelectric material an electric potential gradient causes straining, while stress causes an electric potential in the material (*Piezoelectric Analysis*). This coupling is provided by defining the piezoelectric and dielectric coefficients of a material and can be used in natural frequency extraction, transient dynamic analysis, both linear and nonlinear static stress analysis, and steady-state dynamic analysis procedures. In all procedures, including nonlinear statics and dynamics, the piezoelectric behavior is always assumed to be linear.

Steady Electrical Conduction Procedures

The following electrical conduction analyses procedures are available in Abaqus/Standard:

Coupled thermal-electrical analysis

The electric potential and temperature fields can be solved simultaneously by performing a coupled thermal-electrical analysis (*Coupled Thermal-Electrical Analysis*). In these problems the energy dissipated

by an electrical current flowing through a conductor is converted into thermal energy, and the electrical conductivity can, in turn, be temperature dependent. Thermal loads can be applied, but deformation of the structure is not considered. Coupled thermal-electrical problems can be linear or nonlinear.

Fully coupled thermal-electrical-structural analysis

A coupled thermal-electrical-structural analysis is used to solve simultaneously for the stress/displacement, the electric potential, and the temperature fields. A coupled analysis is used when the thermal, electrical, and mechanical solutions affect each other strongly. An example of such a process is resistance spot welding, where two or more metal parts are joined by fusion at discrete points at the material interface. The fusion is caused by heat generated due to the current flow at the contact points, which depends on the pressure applied at these points.

These problems can be transient or steady state and linear or nonlinear. Cavity radiation effects cannot be included in a fully coupled thermal-electrical-structural analysis. See *Fully Coupled Thermal-Electrical-Structural Analysis* for more details.

Magnetostatic Procedure

The following magnetostatic analysis procedure is available in Abaqus/Standard:

Magnetostatic analysis

A magnetostatic analysis is used to solve for the magnetic vector potential, from which the magnetic field is computed in the entire domain. For example, the magnetic field due to the flow of direct current can be modeled. The procedure supports linear as well as nonlinear magnetic material properties. See *Magnetostatic Analysis* for more details.

Electromagnetic Procedures

Electromagnetic analyses are used to solve for the magnetic vector potential, from which both electric and magnetic fields are computed in the entire domain. The following electromagnetic analysis procedures are available in Abaqus/Standard:

Time-harmonic eddy current analysis

This procedure assumes time-harmonic excitation and response. It supports linear electrical conductivity and linear magnetic material behavior. For example, eddy currents induced in a workpiece that is in the vicinity of a source of excitation (such as a coil carrying alternating current) can be modeled. See *Time-Harmonic Analysis* for more details.

Transient eddy current analysis

This procedure assumes general time variation of the excitation and response. It supports linear electrical conductivity and both linear and nonlinear magnetic material behavior. For example, eddy currents induced in a workpiece that is in the vicinity of a source of excitation (such as a coil carrying time-varying current) can be modeled. See *Transient Analysis* for more details.

Piezoelectric Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

Coupled piezoelectric problems:

- are those in which an electric potential gradient causes straining, while stress causes an electric potential gradient in the material;
- are solved using an eigenfrequency extraction, modal dynamic, static, dynamic, or steady-state dynamic procedure;
- require the use of piezoelectric elements and piezoelectric material properties;
- can be performed for continuum problems in one, two, and three dimensions; and
- can be used in both linear and nonlinear analysis (however, in nonlinear analysis the piezoelectric part of the constitutive behavior is assumed to be linear).

References:

- *Piezoelectric Behavior*
- *Defining an Analysis*
- *Electromagnetic Analysis Procedures*
- *Defining a concentrated charge*
- *Defining a surface charge*
- *Defining a body charge*

Piezoelectric Response

The electrical response of a piezoelectric material is assumed to be made up of piezoelectric and dielectric effects:

$$q_i = e^{\varphi}_{ijk}\varepsilon_{jk} + D^{\varphi(\varepsilon)}_{ij}E_j,$$

where

φ

is the electrical potential,

q_i

is the component of the electric flux vector (also known as the electric displacement) in the ith material direction,

e^{φ}_{ijk}

is the piezoelectric stress coupling,

ε_{ij}

is a small-strain component,

$D_{ij}^{\varphi(\varepsilon)}$

is the material's dielectric matrix for a fully constrained material, and

E_i

is the negative of the gradient of the electrical potential along the ith material direction, $-\partial\varphi/\partial x_i$.

Defining piezoelectric and dielectric properties is discussed in *Piezoelectric Behavior*. The theoretical basis of the piezoelectric analysis capability in Abaqus is defined in *Piezoelectric analysis*.

Procedures Available for Piezoelectric Analysis

Piezoelectric analysis can be carried out with the following procedures:

- *Static Stress Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Transient Modal Dynamic Analysis*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*

Initial Conditions

Initial conditions of piezoelectric quantities cannot be specified. See *Initial Conditions* for a description of the initial conditions that can be applied in static or dynamic procedures.

Boundary Conditions

The electric potential at a node (degree of freedom 9) can be prescribed using a boundary condition (see *Boundary Conditions*). Displacement and rotation degrees of freedom can also be prescribed by using boundary conditions as described in the relevant static and dynamic analysis procedure sections.

Boundary conditions can be prescribed as functions of time by referring to amplitude curves (see *Amplitude Curves*).

In an eigenfrequency extraction step (*Natural Frequency Extraction*) involving piezoelectric elements, the electric potential degree of freedom must be constrained at least at one node to remove singularities from the dielectric part of the element operator. If there are multiple disconnected piezoelectric regions in the model, the electric potential degree of freedom must be constrained at least at one node in each of these regions.

Loads

Both mechanical and electrical loads can be applied in a piezoelectric analysis.

Applying Mechanical Loads

The following types of mechanical loads can be prescribed in a piezoelectric analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*.

Applying Electrical Loads

The following types of electrical loads can be prescribed, as described in *Electromagnetic Loads*:

- Concentrated electric charge.
- Distributed surface electric charge and body electric charge.

Loading in Mode-Based and Subspace-Based Procedures

Electrical charge loads should be used only in conjunction with residual modes in the eigenvalue extraction step, due to the "massless" mode effect. Since the electrical potential degrees of freedom do not have any associated mass, these degrees of freedom are essentially eliminated (similar to Guyan reduction or mass condensation) during the eigenvalue extraction. The residual modes represent the static response corresponding to the electrical charge loads, which will adequately represent the potential degree of freedom in the eigenspace.

Predefined Fields

The following predefined fields can be specified in a piezoelectric analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in piezoelectric elements, nodal temperatures can be specified. The specified temperature affects only temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material Options

The piezoelectric coupling matrix and the dielectric matrix are specified as part of the material definition for piezoelectric materials, as described in *Piezoelectric Behavior*. They are relevant only when the material definition is used with coupled piezoelectric elements.

The mechanical behavior of the material can include linear elasticity only (*Linear Elastic Behavior*).

Each material definition can have a material damping coefficient assigned for procedures where damping can be part of the solution. For piezoelectric materials you can specify piezoelectric damping. You can define stiffness proportional viscous and structural damping by providing damping coefficients for the displacement (mechanical), piezoelectric coupling, and dielectric parts of the damping operator. If you specify piezoelectric damping to define stiffness proportional viscous damping, you cannot specify material damping to define stiffness proportional viscous damping, and vice versa. The same applies for stiffness proportional structural damping.

Input File Usage: Use the following option to define stiffness proportional viscous damping for piezoelectric materials:

**PIEZOELECTRIC DAMPING*, BETA

Use the following option to define stiffness proportional structural damping for piezoelectric materials:

**PIEZOELECTRIC DAMPING*, STRUCTURAL

Abaqus/CAE Usage: Piezoelectric damping is not supported in Abaqus/CAE.

Elements

Piezoelectric elements must be used in a piezoelectric analysis (see *Choosing the Appropriate Element for an Analysis Type*). The electric potential, φ, is degree of freedom 9 at each node of these elements. In addition, regular stress/displacement elements can be used in parts of the model where piezoelectric effects do not need to be considered.

Output

The following output variables are applicable to the electrical solution in a piezoelectric analysis:

Element integration point variables:

EENER Electrostatic energy density.

EPG Magnitude and components of the negative of the electrical potential gradient vector, $-\partial\varphi/\partial\mathbf{x}$.

EPGM Magnitude of the electrical potential gradient vector.

EPG*n* Component *n* of the negative of the electrical potential gradient vector (*n*=1, 2, 3).

EFLX Magnitude and components of the electrical flux (displacement) vector, $\mathbf{q}$.

EFLXM Magnitude of the electrical flux (displacement) vector.

EFLX*n* Component *n* of the electrical flux (displacement) vector (*n*=1, 2, 3).

Whole element variables:

CHRGS Values of distributed electrical charges.

ELCTE Total electrostatic energy in the element, $\int_v \mathbf{EENER}\ dv$.

Nodal variables:

EPOT Electrical potential degree of freedom at a node.

RCHG Reactive electrical nodal charge (conjugate to prescribed electrical potential).

CECHG Concentrated electrical nodal charge.

Limitations

Abaqus does not account for piezoelectric effects in the total energy balance equation, which can lead to an apparent imbalance of the total energy of the model in some situations. For example, if a piezoelectric truss is fixed at one end point and subjected to a potential difference between its two end points, it deforms due to the piezoelectric effect. Subsequently if the truss is held fixed in this deformed configuration and the potential difference removed, strain energy is generated due to the constraints. This results in an equivalent increase in the total energy of the model.

Input File Template

```
*HEADING
…
*MATERIAL, NAME=matl
*ELASTIC
Data lines to define linear elasticity
*PIEZOELECTRIC
Data lines to define piezoelectric behavior
*PIEZOELECTRIC DAMPING, BETA
Data lines to define piezoelectric damping
*DAMPING, ALPHA=α_R
*DIELECTRIC
Data lines to define dielectric behavior

…
*AMPLITUDE, NAME=name
Data lines to define amplitude curve for defining concentrated electric charge
**
*STEP, (optionally NLGEOM)
*STATIC
** or *DYNAMIC, *FREQUENCY, *MODAL DYNAMIC,
** *STEADY STATE DYNAMICS (, DIRECT or , SUBSPACE PROJECTION)
*BOUNDARY
Data lines to define boundary conditions on electrical potential and
displacement (rotation) degrees of freedom
*CECHARGE, AMPLITUDE=name
Data lines to define time-dependent concentrated electric charges
*DECHARGE and/or *DSECHARGE
Data lines to define distributed electric charges
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to define mechanical loading
*END STEP
```

Coupled Thermal-Electrical Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

Coupled thermal-electrical problems:

- are those in which coupling between the electrical potential and temperature fields make it necessary to solve both fields simultaneously;
- require the use of coupled thermal-electrical elements, although pure heat transfer elements can also be used in the model;
- can include a specification of the fraction of electrical energy that will be released as heat;
- can include thermal interactions such as gap radiation, gap conductance, and heat generation between surfaces (see *Thermal Contact Properties*);
- can include cavity radiation effects (see *Cavity Radiation in Abaqus/Standard*);
- can include electrical interactions such as electrical current flowing across surfaces (see *Electrical Contact Properties*);
- allow for transient or steady-state thermal solutions and for steady-state electrical solutions; and
- can be linear or nonlinear.

References:

- *Defining an Analysis*
- *Electromagnetic Analysis Procedures*
- *Electrical Conductivity*
- **COUPLED THERMAL-ELECTRICAL*
- **JOULE HEAT FRACTION*
- *Specifying a joule heat fraction*
- *Configuring a fully coupled, simultaneous heat transfer and electrical procedure*

Coupled Thermal-Electrical Analysis

Joule heating arises when the energy dissipated by an electrical current flowing through a conductor is converted into thermal energy. Abaqus/Standard provides a fully coupled thermal-electrical procedure for analyzing this type of problem: the coupled thermal-electrical equations are solved simultaneously for both temperature and electrical potential at the nodes.

The capability includes the analysis of the electrical problem, the thermal problem, and the coupling between the two problems. Coupling arises from two sources:

- temperature-dependent electrical conductivity or temperature-dependent electrical resistivity, and
- internal heat generation, which is a function of the electrical current density.

The thermal part of the problem can include heat conduction and heat storage (*About Thermal Properties*) as well as cavity radiation effects (*Cavity Radiation in Abaqus/Standard*). Forced convection caused by fluid flowing through the mesh is not considered.

The thermal-electrical equations are unsymmetric; therefore, the unsymmetric solver is invoked automatically if you request coupled thermal-electrical analysis. For problems where coupling between the thermal and electrical solutions is weak or where a pure electrical conduction analysis is required for the entire model, the unsymmetric terms resulting from the interfield coupling may be small or zero. In these problems you can invoke the less costly symmetric storage and solution scheme by solving the thermal and electrical equations separately. The separated technique uses the symmetric solver by default. The thermal-electrical solution schemes are discussed below.

The theoretical basis of coupled thermal-electrical analysis is described in detail in *Coupled thermal-electrical analysis*.

Governing Electric Field Equation

The electric field in a conducting material is governed by Maxwell's equation of conservation of charge. Assuming steady-state direct current, the equation reduces to

$$\int_S \mathbf{J} \cdot \mathbf{n}\, dS = \int_V r_c\, dV,$$

where V is any control volume whose surface is S, $\mathbf{n}$ is the outward normal to S, $\mathbf{J}$ is the electrical current density (current per unit area), and r_c is the internal volumetric current source per unit volume.

The flow of electrical current is described by Ohm's law:

$$\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E} = -\boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}},$$

where

$\mathbf{E}(\mathbf{x})$

is the electrical field intensity, defined as the negative of the gradient of the electrical potential $\mathbf{E} = -\partial\varphi/\partial\mathbf{x}$,

φ

is the electrical potential,

$\boldsymbol{\sigma}^E(\theta, f^\alpha)$

is the electrical conductivity matrix,

θ

is the temperature, and

$f^\alpha, \alpha = 1, 2...$

are predefined field variables.

Using Ohm's law in the conservation equation, written in variational form, provides the governing equation of the finite element model:

$$\int_V \frac{\partial \delta\varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}}\, dV = \int_S \delta\varphi\, J\, dS + \int_V \delta\varphi\, r_c dV,$$

where $J \overset{\text{def}}{=} -\mathbf{J} \cdot \mathbf{n}$ is the current density entering the control volume across S.

Defining the Electrical Conductivity

The electrical conductivity, $\boldsymbol{\sigma}^E$, can be isotropic, orthotropic, or fully anisotropic (see *Electrical Conductivity*). Ohm's law assumes that the electrical conductivity is independent of the electrical field, $\mathbf{E}$. The coupled thermal-electrical problem is nonlinear when the electrical conductivity depends on temperature. The (electrical) conduction properties of the material may alternatively be described in terms of the electrical resistivity of the material, as described below.

Defining the Electrical Resistivity

Alternatively, the electrical conduction properties of the material can be defined in terms of the electrical resistivity, $\boldsymbol{\rho}^E$, which is the inverse of the electrical conductivity, $\boldsymbol{\sigma}^E$. The electrical resistivity can be isotropic, orthotropic, or fully anisotropic (see *Electrical Resistivity*). Ohm's law assumes that the electrical resistivity is independent of the electrical field, $\mathbf{E}$. The coupled thermal-electrical problem is nonlinear when the electrical resistivity depends on temperature.

Specifying the Amount of Thermal Energy Generated due to Electrical Current

Joule's law describes the rate of electrical energy, P_{ec}, dissipated by current flowing through a conductor as

$$P_{ec} = \mathbf{J} \cdot \mathbf{E} = \frac{\partial \varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}}.$$

The amount of this energy released as internal heat within the body is $\eta_v P_{ec}$, where η_v is an energy conversion factor. You specify η_v in the material definition. It is assumed that all the electrical energy is converted into heat ($\eta_v = 1.0$) if you do not include the Joule heat fraction in the material description. The fraction given can include a unit conversion factor, if required.

Input File Usage: **JOULE HEAT FRACTION*

Abaqus/CAE Usage: Property module: material editor: **Thermal->Joule Heat Fraction**

Steady-State Analysis

Steady-state analysis provides the steady-state solution directly. Steady-state thermal analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. Only direct current is considered in the electrical problem, and it is assumed that the system has negligible capacitance. (Electrical transient effects are so rapid that they can be neglected.)

Input File Usage: **COUPLED THERMAL-ELECTRICAL*, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Steady state**

Assigning a "Time" Scale to the Analysis

A steady-state analysis has no intrinsic physically meaningful time scale. Nevertheless, you can assign a "time" scale to the analysis step, which is often convenient for output identification and for specifying prescribed temperatures, electrical potential, and fluxes (heat flux and current density) with varying magnitudes. Thus,

when steady-state analysis is chosen, you specify a "time" period and "time" incrementation parameters for the step; Abaqus/Standard then increments through the step accordingly.

Any fluxes or boundary condition changes to be applied during a steady-state step should be given using appropriate amplitude references to specify their "time" variations (*Amplitude Curves*). If fluxes and boundary conditions are specified for the step without amplitude references, they are assumed to change linearly with "time" during the step—from their magnitudes at the end of the previous step (or zero, if this is the beginning of the analysis) to their newly specified magnitudes at the end of this step (see *Defining an Analysis*).

Transient Analysis

Alternatively, the thermal portion of the coupled thermal-electrical problem can be considered transient. As in steady-state analysis, electrical transient effects are neglected. See *Uncoupled Heat Transfer Analysis* for a more detailed description of the heat transfer capability in Abaqus/Standard.

Input File Usage: **COUPLED THERMAL-ELECTRICAL*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**

Time Incrementation

Time integration in the transient heat transfer problem is done with the same backward Euler method used in uncoupled heat transfer analysis. This method is unconditionally stable for linear problems. You can specify the time increments directly, or Abaqus can select them automatically based on a user-prescribed maximum nodal temperature change in an increment. Automatic time incrementation is generally preferred.

Automatic Incrementation

The time increment size can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that these values are not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see *Time Integration Accuracy in Transient Problems*).

Input File Usage: **COUPLED THERMAL-ELECTRICAL*, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed Incrementation

If you select fixed time incrementation and do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will then be used throughout the analysis.

Input File Usage: **COUPLED THERMAL-ELECTRICAL*
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size: Δt_0**

Spurious Oscillations due to Small Time Increments

In transient heat transfer analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta \ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Ending a Transient Analysis

By default, a transient analysis will end when the specified time period has been completed. Alternatively, you can specify that the analysis should continue until steady-state conditions are reached. Steady state is defined by the temperature change rate; when the temperature changes at a rate that is less than the user-specified rate (given as part of the step definition), the analysis terminates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

**COUPLED THERMAL-ELECTRICAL*, END=PERIOD (default)

Use the following option to end the analysis based on the temperature change rate:

**COUPLED THERMAL-ELECTRICAL*, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **End step when temperature change is less than**

Fully Coupled Solution Schemes

Abaqus/Standard offers an exact as well as an approximate implementation of Newton's method for coupled thermal-electrical analysis.

Exact Implementation

An exact implementation of Newton's method involves a nonsymmetric Jacobian matrix as is illustrated in the following matrix representation of the coupled equations:

$$\begin{bmatrix} K_{\varphi\varphi} & K_{\varphi\theta} \\ K_{\theta\varphi} & K_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta\varphi \\ \Delta\theta \end{Bmatrix} = \begin{Bmatrix} R_\varphi \\ R_\theta \end{Bmatrix},$$

where $\Delta\varphi$ and $\Delta\theta$ are the respective corrections to the incremental electrical potential and temperature, K_{ij} are submatrices of the fully coupled Jacobian matrix, and R_φ and R_θ are the electrical and thermal residual vectors, respectively.

Solving this system of equations requires the use of the unsymmetric matrix storage and solution scheme. Furthermore, the electrical and thermal equations must be solved simultaneously. The method provides quadratic convergence when the solution estimate is within the radius of convergence of the algorithm. The exact implementation is used by default.

Approximate Implementation

Some problems require a fully coupled analysis in the sense that the electrical and thermal solutions evolve simultaneously, but with a weak coupling between the two solutions. In other words, the components in the off-diagonal submatrices $K_{\varphi\theta}, K_{\theta\varphi}$ are small compared to the components in the diagonal submatrices $K_{\varphi\varphi}, K_{\theta\theta}$. For these problems a less costly solution may be obtained by setting the off-diagonal submatrices to zero, so that we obtain an approximate set of equations:

$$\begin{bmatrix} K_{\varphi\varphi} & 0 \\ 0 & K_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta\varphi \\ \Delta\theta \end{Bmatrix} = \begin{Bmatrix} R_\varphi \\ R_\theta \end{Bmatrix}.$$

As a result of this approximation the electrical and thermal equations can be solved separately, with fewer equations to consider in each subproblem. The savings due to this approximation, measured as solver time per iteration, will be of the order of a factor of two, with similar significant savings in solver storage of the factored stiffness matrix. Further, in situations without strong thermal loading due to cavity radiation, the subproblems may be fully symmetric or approximated as symmetric, so that the less costly symmetric storage and solution scheme can be used. The solver time savings for a symmetric solution is an additional factor of two. Unless you explicitly select the unsymmetric solver for the step (*Defining an Analysis*), the symmetric solver will be used with this separated technique.

This modified form of Newton's method does not affect solution accuracy since the fully coupled effect is considered through the residual vector R_j at each increment in time. However, the rate of convergence is no longer quadratic and depends strongly on the magnitude of the coupling effect, so more iterations are generally needed to achieve equilibrium than with the exact implementation of Newton's method. When the coupling is significant, the convergence rate becomes very slow and may prohibit the attainment of a solution. In such cases the exact implementation of Newton's method is required. In cases where it is possible to use this approximation, the convergence in an increment will depend strongly on the quality of the first guess to the incremental solution, which you can control by selecting the extrapolation method used for the step (see *Defining an Analysis*).

Input File Usage: Use the following option to specify a separated solution scheme:

**SOLUTION TECHNIQUE*, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Other**: **Solution technique: Separated**

Uncoupled Electric Conduction and Heat Transfer Analysis

The coupled thermal-electrical procedure can also be used to perform uncoupled electric conduction analysis for the whole model or just part of the model (using coupled thermal-electrical elements). Uncoupled electrical analysis is available by omitting the thermal properties from the material description, in which case only the electric potential degrees of freedom are activated in the element and all heat transfer effects are ignored. If heat transfer effects are ignored in the entire model, you should invoke the separated solution technique described

above. Use of this technique will then invoke the symmetric storage and solution scheme, which is an exact representation of a purely electrical problem.

Similarly, coupled thermal-electrical elements can be used in an uncoupled heat transfer analysis (*Uncoupled Heat Transfer Analysis*), in which case all electric conduction effects are ignored. This feature is useful if a thermal-electrical analysis is followed by a pure heat conduction analysis. A typical example is a welding process, where the electric current is applied instantaneously, followed by a cooldown period during which no electrical effects need to be considered. The symmetric solver is activated by default in an uncoupled heat transfer analysis.

Cavity Radiation

Cavity radiation can be activated in a heat transfer step. This feature involves interacting heat transfer between all of the facets of the cavity surface, dependent on the facet temperatures, facet emissivities, and the geometric view factors between each facet pair. When the thermal emissivity is a function of temperature or field variables, you can specify the maximum allowable emissivity change during an increment in addition to the maximum temperature change to control the time incrementation. See *Cavity Radiation in Abaqus/Standard* for more information.

Input File Usage: Use the following option in the step definition to activate cavity radiation:

**RADIATION VIEW FACTOR*

Use the following option to specify the maximum allowable emissivity change:

**HEAT TRANSFER*, MXDEM=*max_delta_emissivity*

Abaqus/CAE Usage: You can specify the maximum allowable emissivity change for a heat transfer step.

Step module: **Create Step**: **General**: **Heat transfer**: **Incrementation**: **Max. allowable emissivity change per increment**

Initial Conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures or field variables (see *Initial Conditions*). Since only steady-state electrical currents are considered, the initial value of the electrical potential is not relevant.

Boundary Conditions

Boundary conditions can be used to prescribe the electrical potential, $\varphi = \varphi(\mathbf{x}, t)$ (degree of freedom 9), and the temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11), at the nodes. See *Boundary Conditions*.

Boundary conditions can be specified as functions of time by referring to amplitude curves (see *Amplitude Curves*).

A boundary without any prescribed boundary conditions corresponds to an insulated surface.

Loads

Both thermal and electrical loads can be applied in a coupled thermal-electrical analysis.

Applying Thermal Loads

The following types of thermal loads can be prescribed in a coupled thermal-electrical analysis, as described in *Thermal Loads*:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Average-temperature radiation conditions.
- Convective film conditions and radiation conditions.

Applying Electrical Loads

The following types of electrical loads can be prescribed, as described in *Electromagnetic Loads*:

- Concentrated current.
- Distributed surface current densities and body current densities.

Predefined Fields

You can define initial temperature fields in coupled thermal-electrical analyses; other predefined temperature fields are not allowed. Instead, you should use boundary conditions to specify temperatures, as described above.

You can specify other predefined field variables in a coupled thermal-electrical analysis. These values affect only field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

Both thermal and electrical properties are active in coupled thermal-electrical analyses. If thermal properties are omitted, an uncoupled electrical analysis will be performed.

All mechanical behavior material models (such as elasticity and plasticity) are ignored in a coupled thermal-electrical analysis.

Thermal Material Properties

For the heat transfer portion of the analysis, the thermal conductivity must be defined (see *Conductivity*). The specific heat must also be defined for transient heat transfer problems (see *Specific Heat*). If changes in internal energy due to phase changes are important, latent heat can be defined (see *Latent Heat*). Thermal expansion coefficients (*Thermal Expansion*) are not meaningful in a coupled thermal-electrical analysis since deformation of the structure is not considered. Internal heat generation can be specified (see *Uncoupled Heat Transfer Analysis*).

Electrical Material Properties

For the electrical portion of the analysis, either the electrical conductivity (see *Electrical Conductivity*) or the electrical resistivity (see *Electrical Resistivity*) must be defined. The electrical conductivity or the electrical resistivity can be a function of temperature and user-defined field variables. The fraction of electrical energy dissipated as heat can also be defined, as explained above.

Elements

The simultaneous solution in a coupled thermal-electrical analysis requires the use of elements that have both temperature (degree of freedom 11) and electrical potential (degree of freedom 9) as nodal variables. The finite element model can also include pure heat transfer elements (so that a pure heat transfer analysis is provided for that part of the model) and coupled thermal-electrical elements for which no thermal properties are given (so that a pure electrical conduction solution is provided for that part of the model).

Coupled thermal-electrical elements are available in Abaqus/Standard in one dimension, two dimensions (planar and axisymmetric), and three dimensions. See *Choosing the Appropriate Element for an Analysis Type*.

Output

The following output variables can be used to request output relating to the electric conduction solution:

Element integration point variables:

EPG Magnitude and components of the electrical potential gradient vector, $\partial\varphi/\partial\mathbf{x}$.

EPGM Magnitude of the electrical potential gradient vector.

EPG*n* Component *n* of the electrical potential gradient vector (*n*=1, 2, 3).

ECD Magnitude and components of the electrical current density vector, *J*.

JENER Electrical energy dissipated due to flow of current, $P_{ec}t_{step}$.

Whole element variables:

ECURS Distributed applied electrical current.

NCURS Electrical current at nodes due to electric conduction.

ELJD Total electrical energy dissipated due to flow of current, $\int_v P_{ec}t_{step}\,dv$.

Nodal variables:

EPOT Electrical potential, φ.

RECUR Reactive electrical current.

CECUR Concentrated applied electrical current.

Whole model variables:

ALLJD Electrical energy summed over the model.

Surface interaction variables (see *Electrical Contact Properties*):

ECD	Electrical current density.
ECDA	ECD multiplied by area.
ECDT	Time integrated ECD.
ECDTA	Time integrated ECDA.
SJD	Heat flux per unit area generated by the electrical current.
SJDA	SJD multiplied by area.
SJDT	Time integrated SJD.
SJDTA	Time integrated SJDA.
WEIGHT	Heat distribution between interface surfaces, *f*.

Considerations for Steady-State Coupled Thermal-Electrical Analysis

In a steady-state coupled thermal-electrical analysis the electrical energy dissipated due to flow of electrical current at an integration point (output variable JENER) is computed using the following relationship:

$$E_{ec} = P_{ec} t_{step},$$

where E_{ec} denotes the electrical energy dissipated due to flow of electrical current and t_{step} is the current step time. In the above relationship it is assumed that the rate of the electrical energy dissipation, P_{ec}, has a constant value in the step that is equal to the value currently computed.

The output variable JENER and the derived output variables ELJD and ALLJD contain the values of electrical energies dissipated in the current step only. Similarly, the contribution from the electrical current flow to the output variable ALLWK includes only the external work performed in the current step.

Input File Template

```
*HEADING
…
*MATERIAL, NAME=mat1
*CONDUCTIVITY
Data lines to define thermal conductivity
*ELECTRICAL CONDUCTIVITY
Data lines to define electrical conductivity
*JOULE HEAT FRACTION
Data lines to define the fraction of electric energy released as heat
**
*STEP
*COUPLED THERMAL-ELECTRICAL
Data line to define incrementation and steady state
*BOUNDARY
Data lines to define boundary conditions on electrical potential and
temperature degrees of freedom
*CECURRENT
```

Data lines to define concentrated currents
**DECURRENT* and/or **DSECURRENT*
Data lines to define distributed current densities
**CFLUX* and/or **DFLUX* and/or **DSFLUX*
Data lines to define thermal loading
**FILM* and/or **SFILM* and/or **RADIATE* and/or **SRADIATE*
Data lines to define convective film and radiation conditions
…
**CONTACT PRINT* or **CONTACT FILE*
Data lines to request output of surface interaction variables
**END STEP*

Fully Coupled Thermal-Electrical-Structural Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A fully coupled thermal-electrical-structural analysis:

- is performed when coupling between the displacement, temperature, and electrical potential fields makes it necessary to obtain solutions for all three fields simultaneously;
- requires the existence of elements with displacement, temperature, and electrical potential degrees of freedom in the model;
- allows for transient or steady-state thermal solutions, static displacement solutions, and steady-state electrical solutions;
- can include thermal interactions such as gap radiation, gap conductance, and gap heat generation between surfaces (see *Thermal Contact Properties*);
- can include electrical interactions such as gap electrical conductance (see *Electrical Contact Properties*);
- cannot include cavity radiation effects but may include radiation boundary conditions (see *Thermal Loads*);
- takes into account temperature dependence of material properties only for the properties that are assigned to elements with temperature degrees of freedom;
- neglects inertia effects; and
- can be transient or steady state.

References:

- *Defining an Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Coupled Thermal-Electrical Analysis*
- **COUPLED TEMPERATURE-DISPLACEMENT*
- *Configuring a fully coupled, simultaneous heat transfer, electrical, and structural procedure*

Fully Coupled Thermal-Electrical-Structural Analysis

A fully coupled thermal-electrical-structural analysis is the union of a coupled thermal-displacement analysis (see *Fully Coupled Thermal-Stress Analysis*) and a coupled thermal-electrical analysis (see *Coupled Thermal-Electrical Analysis*).

Coupling between the temperature and electrical degrees of freedom arises from temperature-dependent electrical conductivity or temperature-dependent electrical resistivity, temperature-dependent piezoresistivity (if piezoresistive effect is modeled), and internal heat generation (Joule heating), which is a function of the electrical current density. The thermal part of the problem can include heat conduction and heat storage (*About Thermal Properties*). Forced convection caused by fluid flowing through the mesh is not considered.

Coupling between the temperature and displacement degrees of freedom arises from temperature-dependent material properties, thermal expansion, and internal heat generation, which is a function of inelastic deformation of the material. In addition, contact conditions exist in some problems where the heat conducted between surfaces may depend strongly on the separation of the surfaces and/or the pressure transmitted across the surfaces as well as friction (see *About Mechanical Contact Properties* and *Thermal Contact Properties*).

Coupling between the electrical and displacement degrees of freedom arises in problems where electricity flows between contact surfaces. The electrical conduction may depend strongly on the separation of the surfaces and/or the pressure transmitted across the surfaces (see *Electrical Contact Properties*). Additional coupling arises in materials that include the piezoresistive effect, which describes the relative changes of electrical resistivity with stress (see *Piezoresistive Effect*).

An example of a simulation that requires a fully coupled thermal-electrical-structural analysis is resistance spot welding. In a typical spot welding process two or more thin metal sheets are pinched between two electrodes. A large current is passed between the electrodes, which melts the metal between the electrodes and forms a weld. The integrity of the weld depends on many parameters including the electrical conductance between the sheets (which can be a function of contact pressure and temperature).

Fully Coupled Solution Scheme

The coupled thermal-electrical-structural analysis in Abaqus uses an exact implementation of Newton's method and involves a nonsymmetric Jacobian matrix, as illustrated in the following matrix representation of the coupled equations:

$$\begin{bmatrix} K_{uu} & K_{u\varphi} & K_{u\theta} \\ K_{\varphi u} & K_{\varphi\varphi} & K_{\varphi\theta} \\ K_{\theta u} & K_{\theta\varphi} & K_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta\varphi \\ \Delta\theta \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_\varphi \\ R_\theta \end{Bmatrix},$$

where Δu, $\Delta\varphi$, and $\Delta\theta$ are the respective corrections to the displacement, electric potential, and temperature; K_{ij} are submatrices of the fully coupled Jacobian matrix; and R_u, R_φ, and R_θ are the mechanical, electrical, and thermal residuals, respectively.

Solving this system requires the use of the unsymmetric matrix storage and solution scheme. In addition, the mechanical, thermal, and electrical conduction equations must be solved simultaneously. The method provides quadratic convergence when the solution estimate is within the radius of convergence of the algorithm.

Steady-State Analysis

Steady-state analysis provides the steady-state solution directly. Steady-state thermal analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. A static displacement solution is assumed. Only direct current is considered in the electrical problem, and it is assumed that the system has negligible capacitance. Electrical transient effects are so rapid that they can be neglected.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Steady state**

Assigning a "Time" Scale to the Analysis

In steady-state cases you should assign an arbitrary "time" scale to the step: you specify a "time" period and "time" incrementation parameters. This time scale is convenient for changing loads and boundary conditions through the step and for obtaining solutions to highly nonlinear (but steady-state) cases; however, for the latter purpose, transient analysis often provides a natural way of coping with the nonlinearity.

Accounting for Frictional Slip Heat Generation

Frictional slip heat generation is normally neglected in the steady-state case. However, it can still be accounted for if user subroutine *FRIC* provides the incremental frictional dissipation through the variable `SFD`. If frictional

heat generation is present, the heat flux into the two contact surfaces depends on the slip rate of the surfaces. The "time" scale in this case cannot be described as arbitrary, and a transient analysis should be performed.

Transient Analysis

Alternatively, you can perform a transient coupled thermal-electrical-structural analysis. As in steady-state analysis, electrical transient effects are neglected and a static displacement solution is assumed. You can control the time incrementation in a transient analysis directly, or Abaqus/Standard can control it automatically. Automatic time incrementation is generally preferred.

Automatic Incrementation Controlled by a Maximum Allowable Temperature Change

The time increments can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see *Time Integration Accuracy in Transient Problems*).

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed Incrementation

If you do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will be used throughout the analysis, except when the explicit creep integration scheme is used. In this case Abaqus/Standard might decrease the time increment if the stability limit is exceeded.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious Oscillations due to Small Time Increments

In transient analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta\ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta\ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Automatic Incrementation Controlled by the Creep Response

The accuracy of the integration of time-dependent (creep) material behavior is governed by the user-specified accuracy tolerance parameter, $tolerance \geq \left(\dot{\bar{\varepsilon}}^{cr}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}_{t}\right)\Delta t$. This parameter is used to prescribe the maximum strain rate change allowed at any point during an increment, as described in *Rate-Dependent Plasticity: Creep and Swelling*. The accuracy tolerance parameter can be specified together with the maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$ (described above); however, specifying the accuracy tolerance parameter activates automatic incrementation even if $\Delta\theta_{max}$ is not specified.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, DELTMX=$\Delta\theta_{max}$, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle on **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Selecting Explicit Creep Integration

Nonlinear creep problems (*Rate-Dependent Plasticity: Creep and Swelling*) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required as long as no other nonlinearities are present. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For most coupled thermal-electrical-structural analyses, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases the implicit integration scheme may be invoked automatically by Abaqus/Standard.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine *CREEP*; you can restrict Abaqus/Standard to using this method for creep problems (with or without geometric nonlinearity included). See *Rate-Dependent Plasticity: Creep and Swelling* for further details.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle on **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*, **Creep/swelling/viscoelastic integration: Explicit**

Excluding Creep and Viscoelastic Response

You can specify that no creep or viscoelastic response occurs during a step even if creep or viscoelastic material properties have been defined.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, DELTMX=$\Delta\theta_{max}$, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable Problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers a method of stabilizing this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Units

In coupled problems where two or three different fields are active, take care when choosing the units of the problem. If the choice of units is such that the terms generated by the equations for each field are different by many orders of magnitude, the precision on some computers may be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid ill-conditioned matrices. For example, consider using units of megapascal (MPa) instead of pascal (Pa) for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations, the heat flux continuity equations, and the conservation of charge equations.

Initial Conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures. Initial stresses, field variables, etc. can also be defined; *Initial Conditions* describes all of the initial conditions that are available for a fully coupled thermal-electrical-structural analysis.

Boundary Conditions

Boundary conditions can be used to prescribe temperatures (degree of freedom 11), displacements/rotations (degrees of freedom 1–6), or electrical potentials (degree of freedom 9) at nodes in a fully coupled thermal-electrical-structural analysis (see *Boundary Conditions*).

Boundary conditions can be specified as functions of time by referring to amplitude curves (*Amplitude Curves*).

Loads

The following types of thermal loads can be prescribed in a fully coupled thermal-electrical-structural analysis, as described in *Thermal Loads*:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Node-based film and radiation conditions.
- Average-temperature radiation conditions.
- Element and surface-based film and radiation conditions.

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*.

The following types of electrical loads can be prescribed, as described in *Electromagnetic Loads*:

- Concentrated current.
- Distributed surface current densities and body current densities.

Predefined Fields

You can define initial temperature fields in fully coupled thermal-electrical-structural analyses; other predefined temperature fields are not allowed. Instead, you should use boundary conditions to prescribe temperature degree of freedom 11, as described earlier.

You can specify other predefined field variables in a fully coupled thermal-electrical-structural analysis. These values affect only field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

The materials in a fully coupled thermal-electrical-structural analysis must have thermal properties (such as conductivity), mechanical properties (such as elasticity), and electrical properties (such as electrical conductivity or electrical resistivity) defined. In addition, for a piezoresistive material the piezoresistivity coefficients must be defined to model the changes of the relative electrical resistivity with stress. See *Abaqus Materials Guide* for details on the material models available in Abaqus.

Internal heat generation can be specified; see *Uncoupled Heat Transfer Analysis*.

Thermal strain will arise if thermal expansion (*Thermal Expansion*) is included in the material property definition.

A fully coupled thermal-electrical-structural analysis can be used to analyze static creep and swelling problems, which generally occur over fairly long time periods (*Rate-Dependent Plasticity: Creep and Swelling*); viscoelastic materials (*Time Domain Viscoelasticity*); or viscoplastic materials (*Rate-Dependent Yield*).

Rate-Dependent Yield and Friction

You can control whether to consider or ignore the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient within the step.

Input File Usage: Use the following option to consider rate dependence within the step:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, RATE DEPENDENCE=ON (default)

Use the following option to ignore rate dependence within the step:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTRICAL, RATE DEPENDENCE=OFF

Abaqus/CAE Usage: Abaqus/CAE always considers the strain rate–dependence of the yield stress and the slip rate–dependence of the friction coefficient.

Inelastic Energy Dissipation as a Heat Source

You can specify an inelastic heat fraction in a fully coupled thermal-electrical-structural analysis to provide for inelastic energy dissipation as a heat source. The heat flux per unit volume, r^{pl}, that is added into the thermal energy balance is computed using the equation

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

or, in the case when the nonlinear isotropic/kinematic hardening model is used, from the following equation:

$$r^{pl} = \eta (\boldsymbol{\sigma} - \boldsymbol{\alpha}) : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where η is a user-defined factor (assumed constant), $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\alpha}$ is the backstress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining.

Inelastic heat fractions are typically used in the simulation of high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation significantly influences temperature-dependent material properties. The generated heat is treated as a volumetric heat flux source term in the heat balance equation.

An inelastic heat fraction can be specified for materials with plastic behavior that use the Mises or Hill yield surface (*Inelastic Behavior*), and it can also be used with the combined isotropic/kinematic hardening model. The inelastic heat fraction can be specified for user-defined material behavior in Abaqus/Explicit and is multiplied by the inelastic energy dissipation coded in the user subroutine to obtain the heat flux. In Abaqus/Standard the inelastic heat fraction cannot be used with user-defined material behavior; in this case the heat flux that must be added to the thermal energy balance is computed directly in the user subroutine.

In Abaqus/Standard an inelastic heat fraction can also be specified for hyperelastic material definitions that include time-domain viscoelasticity (*Time Domain Viscoelasticity*).

The default value of the inelastic heat fraction is 0.9. If you do not include the inelastic heat fraction behavior in the material definition, the heat generated by inelastic deformation is not included in the analysis.

Input File Usage: **INELASTIC HEAT FRACTION*
η

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Inelastic Heat Fraction**: **Fraction:** η

Specifying the Amount of Thermal Energy Generated due to Electrical Current

Joule's law describes the rate of electrical energy, P_{ec}, dissipated by current flowing through a conductor as

$$P_{ec} = \mathbf{J} \cdot \mathbf{E} = \frac{\partial \varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^{E} \cdot \frac{\partial \varphi}{\partial \mathbf{x}}.$$

The amount of this energy released as internal heat within the body is $\eta_v P_{ec}$, where η_v is an energy conversion factor. You specify η_v in the material definition. It is assumed that all the electrical energy is converted into heat ($\eta_v = 1.0$) if you do not include the Joule heat fraction in the material description. The fraction given can include a unit conversion factor, if required.

Input File Usage: **JOULE HEAT FRACTION*

Abaqus/CAE Usage: Property module: material editor: **Thermal**->**Joule Heat Fraction**

Elements

Coupled thermal-electrical-structural elements that have displacements, temperatures, and electrical potentials as nodal variables are available. Simultaneous temperature/electrical potential/displacement solution requires the use of such elements; pure displacement and temperature-displacement elements can be used in part of the model in a fully coupled thermal-electrical-structural analysis, but pure heat transfer elements cannot be used.

The first-order coupled thermal-electrical-structural elements in Abaqus use a constant temperature over the element to calculate thermal expansion. The second-order coupled thermal-electrical-structural elements in Abaqus use a lower-order interpolation for temperature than for displacement (parabolic variation of displacements and linear variation of temperature) to obtain a compatible variation of thermal and mechanical strain.

Output

See *Abaqus/Standard Output Variable Identifiers* for a complete list of output variables. The types of output available are described in *About Output*.

Considerations for Steady-State Coupled Thermal-Electrical-Structural Analysis

In a steady-state coupled thermal-electrical-structural analysis the electrical energy dissipated due to flow of electrical current at an integration point (output variable JENER) is computed using the following relationship:

$$E_{ec} = P_{ec} t_{step},$$

where E_{ec} denotes the electrical energy dissipated due to flow of electrical current and t_{step} is the current step time. In the above relationship it is assumed that the rate of the electrical energy dissipation, P_{ec}, has a constant value in the step that is equal to the value currently computed.

The output variable JENER and the derived output variables ELJD and ALLJD contain the values of electrical energies dissipated in the current step only. Similarly, the contribution from the electrical current flow to the output variable ALLWK includes only the external work performed in the current step.

Input File Template

```
*HEADING
…
** Specify the coupled thermal-electrical-structural element type
*ELEMENT, TYPE=Q3D8
…
**
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL
Data line to define incrementation
*BOUNDARY
Data lines to define nonzero boundary conditions on displacement,
temperature or electrical potential degrees of freedom
*CFLUX and/or *CFILM and/or
*CRADIATE and/or *DFLUX and/or
*DSFLUX and/or *FILM and/or
*SFILM and/or *RADIATE and/or
*SRADIATE
Data lines to define thermal loads
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to define mechanical loads
*CECURRENT
```

Data lines to define concentrated currents
**DECURRENT* and/or **DSECURRENT*
Data lines to define distributed current densities
**FIELD*
Data lines to define field variable values
**END STEP*

Eddy Current Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

Eddy current problems:

- involve coupling between electric and magnetic fields, which are solved for simultaneously;
- solve Maxwell's equations describing electromagnetic phenomena under the low-frequency assumption that neglects the effects of displacement currents;
- require the use of electromagnetic elements in the whole domain;
- require that magnetic permeability is specified in the whole domain and electrical conductivity is specified in the conducting regions;
- allow for both time-harmonic and transient electromagnetic solutions;
- allow predefined conductor translation and rotation;
- calculate as output variables, rate of Joule heating and intensity of magnetic body forces associated with eddy currents, and these output variables can be transferred from a time-harmonic electromagnetic solution to drive a subsequent heat transfer, coupled temperature-displacement, or stress/displacement analysis, thereby allowing for the coupling of electromagnetic fields with thermal and/or mechanical fields in a sequentially coupled manner; and
- can be solved using continuum elements in two- and three-dimensional space.

References:

- *Mapping Thermal and Magnetic Loads*
- *Electromagnetic Analysis Procedures*
- *Electrical Conductivity*
- *Magnetic Permeability*
- *Electromagnetic Loads*
- *Predefined Loads for Sequential Coupling*
- *Output to the Output Database*
- **ELECTROMAGNETIC*
- **D EM POTENTIAL*
- **DECURRENT*
- **DSECURRENT*
- **MOTION*
- *UDECURRENT*
- *UDEMPOTENTIAL*
- *UDSECURRENT*
- *Configuring a time-harmonic electromagnetic analysis*
- *Defining a magnetic vector potential boundary condition*

Eddy Current Analysis

Eddy currents are generated in a metal workpiece when it is placed within a time-varying magnetic field. Joule heating arises when the energy dissipated by the eddy currents flowing through the workpiece is converted into thermal energy. This heating mechanism is usually referred to as induction heating; the induction cooker is an example of a device that uses this mechanism. The time-varying magnetic field is usually generated by a coil that is placed close to the workpiece. The coil carries either a known amount of total current or an unknown amount of current under a known potential (voltage) difference. The current in the coil is assumed to be alternating at a known frequency for a time-harmonic eddy current analysis but may have an arbitrary variation in time for a transient eddy current analysis.

The time-harmonic eddy current analysis procedure is based on the assumption that a time-harmonic excitation with a certain frequency results in a time-harmonic electromagnetic response with the same frequency everywhere in the domain. In other words, both the electric and the magnetic fields oscillate at the same frequency as that of the alternating current in the coil. The transient eddy current analysis does not make any assumption regarding the time-variation of the current in the coil; in fact any arbitrary time variation can be specified, and the electric and magnetic fields follow from the solution to Maxwell's equations in the time domain.

The eddy current analysis provides output, such as Joule heat dissipation or magnetic body force intensity, that can be transferred, from a time-harmonic eddy current analysis only, to drive a subsequent heat transfer, coupled temperature-displacement, or stress/displacement analysis. This allows for modeling the interactions of the electromagnetic fields with thermal and/or mechanical fields in a sequentially coupled manner. See *Mapping Thermal and Magnetic Loads* and *Predefined Loads for Sequential Coupling* for details.

Electromagnetic elements must be used to model the response of all the regions in an eddy current analysis including the coil, the workpiece, and the space in between and surrounding them. To obtain accurate solutions, the outer boundary of the space (surrounding the coil and the workpiece) being modeled must be at least a few characteristic length scales away from the device on all sides.

The electromagnetic elements use an element edge-based interpolation of the fields instead of the standard node-based interpolation. The user-defined nodes only define the geometry of the elements; and the degrees of freedom of the element are not associated with these nodes, which has implications for applying boundary conditions (see *Boundary Conditions* below).

Governing Field Equations

The electric and magnetic fields are governed by Maxwell's equations describing electromagnetic phenomena. The formulation is based on the low-frequency assumption, which neglects the displacement current correction term in Ampere's law. This assumption is appropriate when the wavelength of the electromagnetic waves corresponding to the excitation frequency is large compared to typical length scales over which the response is computed. In the following discussion, the governing equations are written for a linear medium.

Time-Harmonic Analysis

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$.

The solution procedure seeks a time-harmonic electromagnetic response, $\mathbf{A}_0 \exp(i\omega t)$, with frequency ω radians/sec when the system is subjected to a time-harmonic excitation of the same frequency; for example, through an impressed oscillating volume current density, $\mathbf{J}_0 \exp(i\omega t)$. In the preceding expressions the vectors $\mathbf{A}_0$ and $\mathbf{J}_0$ represent the amplitudes of the magnetic vector potential and applied volume current density vector, respectively, while the exponential factors (with $i = \sqrt{-1}$) represent the corresponding phases. Under these assumptions, Maxwell's equations in the absence of conductor motion reduce to

$$\nabla \times \left(\mu^{-1} \cdot \nabla \times \mathbf{A}_0\right) + i\omega\sigma^E \cdot \mathbf{A}_0 = \mathbf{J}_0$$

in terms of the amplitudes of the field quantities, $\mathbf{A}_0$ and $\mathbf{J}_0$; the magnetic permeability tensor, $\boldsymbol{\mu}$; and the electrical conductivity tensor, $\boldsymbol{\sigma}^E$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$, while the electrical conductivity relates the volume current density, $\mathbf{J}$, and the electric field, $\mathbf{E}$, by Ohm's law: $\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A}_0 \cdot \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}_0\right)\, dV + i\omega \int_V \delta\mathbf{A}_0 \cdot \boldsymbol{\sigma}^E \cdot \mathbf{A}_0\, dV = \int_V \delta\mathbf{A}_0 \cdot \mathbf{J}_0\, dV + \int_S \delta\mathbf{A}_0 \cdot \mathbf{K}_0\ dS,$$

where $\delta\mathbf{A}_0$ represents the variation of the magnetic vector potential, and $\mathbf{K}_0$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the in-phase (real) and out-of-phase (imaginary) components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential.

Transient Analysis

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, assumed to be a function of spatial position and time, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$. The solution procedure seeks a time-dependent electromagnetic response, $\mathbf{A}(\mathbf{x}, t)$, when the system is subjected to a time-dependent excitation; for example, through an impressed distribution of volume current density, $\mathbf{J}(\mathbf{x}, t)$. Under these assumptions, Maxwell's equations in the absence of conductor motion reduce to

$$\nabla \times \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}\right) + \boldsymbol{\sigma}^E \cdot \partial\mathbf{A}/\partial t = \mathbf{J}$$

in terms of the field quantities, $\mathbf{A}$ and $\mathbf{J}$; the magnetic permeability tensor, $\boldsymbol{\mu}$; and the electrical conductivity tensor, $\boldsymbol{\sigma}^E$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$, while the electrical conductivity relates the volume current density, $\mathbf{J}$, and the electric field, $\mathbf{E}$, by Ohm's law: $\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A} \cdot \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}\right)\, dV + \int_V \delta\mathbf{A} \cdot \boldsymbol{\sigma}^E \cdot \partial\mathbf{A}/\partial t\, dV = \int_V \delta\mathbf{A} \cdot \mathbf{J}\, dV + \int_S \delta\mathbf{A} \cdot \mathbf{K}\ dS,$$

where $\delta\mathbf{A}$ represents the variation of the magnetic vector potential, and $\mathbf{K}$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential.

Predefined Conductor Motion

Electric fields induced in a conductor have two parts: the first part due to the changing magnetic flux (Faraday’s law of induction), which is already accounted for in the formulation above, and the second part due to the motion of the conductor in the magnetic field. The second part modifies the governing equation as follows (shown only for a transient procedure, although the capability is available for the time-harmonic procedure as well):

$$\nabla \times \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}\right) + \boldsymbol{\sigma}^{E} \cdot \partial \mathbf{A} / \partial t - \boldsymbol{\sigma}^{E} \cdot \mathbf{v} \times (\nabla \times \mathbf{A}) = \mathbf{J}$$

in terms of the prescribed velocity of motion $\mathbf{v}$. You can prescribe both translational and rotational velocity of motion. The formulation assumes that the conductor is uniform in the direction of motion; in other words, it does not have any geometric features in the direction of motion.

Conductor motion results in unsymmetric contributions toward the element operator. By default, the unsymmetric storage and solution scheme is used with conductor motion.

Defining the Magnetic Behavior

The magnetic behavior of the electromagnetic medium can be linear or nonlinear. However, only linear magnetic behavior is available for time-harmonic eddy current analysis. Linear magnetic behavior is characterized by a magnetic permeability tensor that is assumed to be independent of the magnetic field. It is defined through direct specification of the absolute magnetic permeability tensor, $\boldsymbol{\mu}$, which can be isotropic, orthotropic, or fully anisotropic (see *Magnetic Permeability*). The magnetic permeability can also depend on temperature and/or predefined field variables. For a time-harmonic eddy current analysis, the magnetic permeability can also depend on frequency.

Nonlinear magnetic behavior, which is available only for transient eddy current analysis, is characterized by magnetic permeability that depends on the strength of the magnetic field. The nonlinear magnetic material model in Abaqus is suitable for ideally soft magnetic materials characterized by a monotonically increasing response in B–H space, where B and H refer to the strengths of the magnetic flux density vector and the magnetic field vector, respectively. Nonlinear magnetic behavior is defined through direct specification of one or more B–H curves that provide B as a function of H and, optionally, temperature and/or predefined field variables, in one or more directions. Nonlinear magnetic behavior can be isotropic, orthotropic, or transversely isotropic (which is a special case of the more general orthotropic behavior).

Defining the Electrical Conductivity

The electrical conductivity, $\boldsymbol{\sigma}^{E}$, can be isotropic, orthotropic, or fully anisotropic (see *Electrical Conductivity*). The electrical conductivity can also depend on temperature and/or predefined field variables. For a time-harmonic eddy current analysis, the electrical conductivity can also depend on frequency. Ohm's law assumes that the electrical conductivity is independent of the electrical field, $\mathbf{E}$.

Time-Harmonic Analysis

The eddy current analysis procedure provides the time-harmonic solution directly at a given excitation frequency. You can specify one or more excitation frequencies, one or more frequency ranges, or a combination of excitation frequencies and ranges.

Input File Usage:

**ELECTROMAGNETIC*, LOW FREQUENCY, TIME HARMONIC
lower_freq1, upper_freq1, num_pts1
lower_freq2, upper_freq2, num_pts2
...
single_freq1
single_freq2
...

For example, the following input illustrates the simplest case of specifying excitation at a single frequency:

**ELECTROMAGNETIC*, LOW FREQUENCY, TIME HARMONIC
single_freq1

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Electromagnetic, Time harmonic**; enter data in table, and add rows as necessary

Transient Analysis

The eddy current analysis procedure provides the transient solution to a given arbitrary time-dependent excitation.

Input File Usage: **ELECTROMAGNETIC*, LOW FREQUENCY, TRANSIENT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Time Incrementation

Time integration in the transient eddy current analysis is done with the backward Euler method. This method is unconditionally stable for linear problems but may lead to inaccuracies if time increments are too large. The resulting system of equations can be nonlinear in general, and Abaqus/Standard uses Newton's method to solve the system. The solution usually is obtained as a series of increments, with iterations to obtain equilibrium within each increment. Increments must sometimes be kept small to ensure accuracy of the time integration procedure. The choice of increment size is also a matter of computational efficiency: if the increments are too large, more iterations are required. Furthermore, Newton's method has a finite radius of convergence; too large an increment can prevent any solution from being obtained because the initial state is too far away from the equilibrium state that is being sought—it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic Incrementation

In most cases the default automatic incrementation scheme is preferred because it will select increment sizes based on computational efficiency. However, you must ensure that the time increments are such that the time integration results in an accurate solution. Abaqus/Standard does not have any built in checks to ensure integration accuracy.

Input File Usage: **ELECTROMAGNETIC*, LOW FREQUENCY, TRANSIENT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Direct Incrementation

Direct user control of the increment size is also provided; if you have considerable experience with a particular problem, you may be able to select a more economical approach.

Input File Usage: **ELECTROMAGNETIC*, LOW FREQUENCY, TRANSIENT, DIRECT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Ill-Conditioning in Eddy Current Analyses with Electrically Nonconductive Regions

In an eddy current analysis it is very common that large portions of the model consist of electrically nonconductive regions, such as air and/or a vacuum. In such cases it is well known that the associated stiffness matrix can be very ill-conditioned; i.e., it can have many singularities (*Bíró, 1999*). Abaqus uses a special iterative solution technique to prevent the ill-conditioned matrix from negatively impacting the computed electric and magnetic

fields. The default implementation works well for many problems. However, there can be situations in which the default numerical scheme fails to converge or results in a noisy solution. In such cases adding a "small" amount of artificial electrical conductivity to the nonconductive domain may help regularize the problem and allow Abaqus to converge to the correct solution. The artificial electrical conductivity should be chosen such that the electromagnetic waves propagating through these regions undergo little modification and, in particular, do not experience the sharp exponential decay that is typical when such fields impinge upon a real conductor. It is recommended that you set the artificial conductivity to be about five to eight orders of magnitude less than that of any of the conductors in the model.

As an alternative to specifying electrical conductivity in the nonconductive domain, Abaqus also provides a stabilization scheme to help mitigate the effects of the ill-conditioning. You can provide input to this stabilization algorithm by specifying the stabilization factor, which is assumed to be 1.0 by default if the stabilization scheme is used. Higher values of the stabilization factor lead to more stabilization, while lower values of the stabilization factor lead to less stabilization.

Input File Usage: Use the following to use stabilization in a time-harmonic procedure:

**ELECTROMAGNETIC*, LOW FREQUENCY, TIME HARMONIC, STABILIZATION=*stabilization factor*

Use the following to use stabilization in a transient procedure:

**ELECTROMAGNETIC*, LOW FREQUENCY, TRANSIENT, STABILIZATION=*stabilization factor*

Abaqus/CAE Usage: The default stabilization factor cannot be modified in Abaqus/CAE.

Prescribed Conductor Motion

You can specify conductor motion by prescribing the direction and magnitude of the translational or rotational velocity vector over an element set representing the conductor. Only a single conductor motion is allowed in a step.

Input File Usage: Use the following to prescribe a translational velocity:

**MOTION*, ELEMENT, TRANSLATION

Use the following to prescribe a rotational velocity:

**MOTION*, ELEMENT, ROTATION

Abaqus/CAE Usage: Specifying conductor motion is not supported in Abaqus/CAE.

Initial Conditions

Initial values of temperature and/or predefined field variables can be specified. These values affect only temperature and/or field-variable-dependent material properties, if any. Initial conditions on the electric and/or magnetic fields cannot be specified in an eddy current analysis.

Boundary Conditions

Electromagnetic elements use an element edge-based interpolation of the fields. The degrees of freedom of the element are not associated with the user-defined nodes, which only define the geometry of the element. Consequently, the standard node-based method of specifying boundary conditions cannot be used with electromagnetic elements. The method used for specifying boundary conditions for electromagnetic elements is described in the following paragraphs.

Boundary conditions in Abaqus typically refer to what are traditionally known as Dirichlet-type boundary conditions in the literature, where the values of the primary variable are known on the whole boundary or on a portion of the boundary. The alternative, Neumann-type boundary conditions, refer to situations where the values of the conjugate to the primary variable are known on portions of the boundary. In Abaqus Neumann-type boundary conditions are represented as surface loads in the finite-element formulation.

For electromagnetic boundary value problems, Dirichlet boundary conditions on an enclosing surface must be specified as $\mathbf{A} \times \mathbf{n}$, where $\mathbf{n}$ is the outward normal to the surface, as discussed in this section. Neumann boundary conditions must be specified as the surface current density vector, $\mathbf{K} = \mathbf{H} \times \mathbf{n}$, as discussed in *Loads* below.

In Abaqus Dirichlet boundary conditions are specified as magnetic vector potential, $\mathbf{A}$, on (element-based) surfaces that represent symmetry planes and/or external boundaries in the model; Abaqus computes $\mathbf{A} \times \mathbf{n}$ for the representative surfaces. In applications where the electromagnetic fields are driven by a current-carrying coil that is close to the workpiece, the model may span a domain that is up to 10 times the characteristic length scale associated with the coil/workpiece assembly. In such cases, the electromagnetic fields are assumed to have decayed sufficiently in the far-field, and the value of the magnetic vector potential can be set to zero in the far-field boundary. On the other hand, in applications such as one where a conductor is embedded in a uniform (but varying time-harmonically in a time-harmonic eddy current analysis or with a more general time variation in a transient eddy current analysis) far-field magnetic field, it may be necessary to specify nonzero values of the magnetic vector potential on some portions of the external boundary. In this case an alternative method to model the same physical phenomena is to specify the corresponding unique value of surface current density, $\mathbf{K}$, on the far-field boundary (see *Loads* below). $\mathbf{K}$ can be computed based on known values of the far-field magnetic field.

A surface without any prescribed boundary condition corresponds to a surface with zero surface currents, or no loads.

Nonuniform boundary conditions can be defined with user subroutine *UDEMPOTENTIAL*.

Prescribing Boundary Conditions in a Time-Harmonic Eddy Current Analysis

In a time-harmonic eddy current analysis the boundary conditions are assumed to be time harmonic and are applied simultaneously to both the real and imaginary parts of the magnetic vector potential. It is not possible to specify Dirichlet boundary conditions on the real parts and Neumann boundary conditions on the imaginary parts and vice versa. Abaqus automatically restrains both the real and imaginary parts even if only one part is prescribed explicitly. The unspecified part is assumed to have a magnitude of zero.

When you prescribe the boundary condition on an element-based surface for a time-harmonic eddy current analysis (see *Element-Based Surface Definition*), you must specify the surface name, the region type label (S), the boundary condition type label, an optional orientation name, the magnitude of the real part of the boundary condition, the direction vector for the real part of the boundary condition, the magnitude of the imaginary part of the boundary condition, and the direction vector for the imaginary part of the boundary condition. The optional orientation name defines the local coordinate system in which the components of the magnetic vector potential are defined. By default, the components are defined with respect to the global directions. The specified direction vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

During a time-harmonic eddy current analysis, frequency-dependent boundary conditions can be prescribed as described in *Frequency-Dependent Boundary Conditions in a Time-Harmonic Eddy Current Analysis* below.

Input File Usage: Use the following option in a time-harmonic eddy current analysis to define both the real (in-phase) and imaginary (out-of-phase) parts of the boundary condition on element-based surfaces:

**D EM POTENTIAL*
surface name, S, bc type label, orientation, magnitude of real part, direction vector of real part, magnitude of imaginary part, direction vector of imaginary part

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Electrical/Magnetic** for the **Category** and **Magnetic vector potential** for the **Types for Selected Step**; **Distribution**: **Uniform** or **User-defined**; *real components + imaginary components*

Prescribing Boundary Conditions in a Transient Eddy Current Analysis

The method of specification of the boundary condition for a transient eddy current analysis is substantially similar to that of the time-harmonic eddy current analysis, except that the concepts of real and imaginary are not relevant any more. In this case you specify the magnitude of the magnetic vector potential, followed by its direction vector. The specified direction vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

During a transient eddy current analysis, prescribed boundary conditions can be varied using an amplitude definition (see *Amplitude Curves*).

Input File Usage: Use the following option in a transient eddy current analysis to define the boundary condition on element-based surfaces:

**D EM POTENTIAL*
surface name, S, bc type label, orientation, magnitude, direction vector

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Abaqus/CAE Usage: Transient eddy current analysis is not supported in Abaqus/CAE.

Frequency-Dependent Boundary Conditions in a Time-Harmonic Eddy Current Analysis

An amplitude definition can be used to specify the amplitude of a boundary condition as a function of frequency (*Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**D EM POTENTIAL*, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name**: *amplitude_name*
Load module: **Create Boundary Condition**: choose **Electrical/Magnetic** for the

Category and **Magnetic vector potential** for the **Types for Selected Step**; **Amplitude**: *amplitude_name*

Loads

The following types of electromagnetic loads can be applied in an eddy current analysis (see *Prescribing Electromagnetic Loads for Eddy Current and/or Magnetostatic Analyses* for details):

- Element-based distributed volume current density vector: $\mathbf{J}_0(\mathbf{x})$ in a time-harmonic eddy current analysis, and $\mathbf{J}(\mathbf{x}, t)$ in a transient eddy current analysis
- Surface-based distributed surface current density vector: $\mathbf{K}_0(\mathbf{x})$ in a time-harmonic eddy current analysis, and $\mathbf{K}(\mathbf{x}, t)$ in a transient eddy current analysis

All loads in a time-harmonic eddy current analysis are assumed to be time-harmonic with the excitation frequency. During a transient eddy current analysis all loads can be varied using an amplitude definition (see *Amplitude Curves*).

Nonuniform loads can be specified using user subroutines *UDECURRENT* and *UDSECURRENT*.

Frequency-Dependent Loading in a Time-Harmonic Eddy Current Analysis

In a time-harmonic eddy current analysis, an amplitude definition can be used to specify the amplitude of a load as a function of frequency (*Amplitude Curves*).

Predefined Fields

Predefined temperature and field variables can be specified in an eddy current analysis; however, user-defined fields that allow the value of field variables at a material point to be redefined via user subroutine *USDFLD* are not supported. These values affect only temperature and/or field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

Magnetic material behavior (see *Magnetic Permeability*) must be specified everywhere in the model. Only linear magnetic behavior is supported in a time-harmonic eddy current analysis, but nonlinear magnetic behavior is also supported in a transient eddy current analysis. Linear magnetic behavior can be defined by specifying the magnetic permeability directly, while nonlinear magnetic behavior is defined in terms of one or more B–H curves. Electrical conductivity (see *Electrical Conductivity*) must be specified in conductor regions. All other material properties are ignored in an eddy current analysis.

Both magnetic permeability and electrical conductivity can be functions of frequency, predefined temperature, and field variables in a time-harmonic eddy current analysis. In a transient eddy current analysis, all material behavior can be functions of predefined temperature and/or field variables.

Permanent magnets (see *Magnetic Permeability*) can be included in a transient eddy current analysis.

Elements

Electromagnetic elements must be used to model all regions in an eddy current analysis. Unlike conventional finite elements, which use node-based interpolation, these elements use edge-based interpolation with the tangential components of the magnetic vector potential along element edges serving as the primary degrees of freedom.

Electromagnetic elements are available in Abaqus/Standard in two dimensions (planar only) and three dimensions (see *Choosing the Appropriate Element for an Analysis Type*). The planar elements are formulated in terms of an in-plane magnetic vector potential, thereby the magnetic flux density and magnetic field vectors only have an out-of-plane component. The electric field and the current density vectors are in-plane for the planar elements.

Output

Eddy current analysis provides output only to the output database (`.odb`) file (see *Output to the Output Database*). Output to the data (`.dat`) file and to the results (`.fil`) file is not available. For the first four vector quantities listed below (which are derived from the magnetic vector potential and the constitutive equations) and the applied volume current density vector, the magnitude and components of the real and imaginary parts are output in a time-harmonic eddy current procedure.

Element centroidal variables:

EMB Magnitude and components of the magnetic flux density vector, $\mathbf{B}$.

EMH Magnitude and components of the magnetic field vector, $\mathbf{H}$.

EME Magnitude and components of the electric field vector, $\mathbf{E}$.

EMCD Magnitude and components of the eddy current density vector, $\sigma\mathbf{E}$, in conducting regions.

EMCDA Magnitude and components of the applied volume current density vector.

EMBF Magnetic body force intensity vector (force per unit volume per unit time) due to flow of induced current.

EMBFC Complex magnetic body force intensity vector (real and imaginary parts of the force per unit volume) due to flow of current. Only available in a time-harmonic eddy current analysis.

EMJH Rate of Joule heating (amount of heat per unit volume per unit time) due to flow of current.

TEMP Temperature at the centroid of the element. For a time-harmonic eddy current analysis, this value represents the temperature that is used for evaluating the temperature-dependent material properties.

Whole element variables:

ELJD Total rate of Joule heating (amount of heat per unit time) due to flow of current in an element.

EVOL Element volume.

Whole model variables:

ALLJD Rate of Joule heating (amount of heat per unit time) summed over the model or an element set.

Input File Template

The following is an input file template that makes use of linear magnetic material behavior in a time-harmonic eddy current analysis:

```
*HEADING
…
*MATERIAL, NAME=mat1
*MAGNETIC PERMEABILITY
Data lines to define magnetic permeability
*ELECTRICAL CONDUCTIVITY
Data lines to define electrical conductivity in the conductor region
**
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
Data line to specify excitation frequencies
*D EM POTENTIAL
Data lines to define boundary conditions on magnetic vector potential
*DECURRENT
Data lines to define element-based distributed volume current density vector
*DSECURRENT
Data lines to define surface-based distributed surface current density vector
*OUTPUT, FIELD or HISTORY
Data lines to request element-based output
*ENERGY OUTPUT
Data line to request whole model Joule heat dissipation output
*END STEP
```

The following is an input file template that makes use of nonlinear magnetic material behavior in a transient eddy current analysis:

```
*HEADING
…
*MATERIAL, NAME=mat1
*MAGNETIC PERMEABILITY, NONLINEAR
*NONLINEAR BH, DIR=direction
Data lines to define nonlinear B-H curve
*ELECTRICAL CONDUCTIVITY
Data lines to define electrical conductivity in the conductor region
**
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT
*D EM POTENTIAL
Data lines to define boundary conditions on magnetic vector potential
*DECURRENT
Data lines to define element-based distributed volume current density vector
*DSECURRENT
Data lines to define surface-based distributed surface current density vector
*OUTPUT, FIELD or HISTORY
Data lines to request element-based output
*ENERGY OUTPUT
Data line to request whole model Joule heat dissipation output
*END STEP
```

References

- Bíró, O., "Edge Element Formulation of Eddy Current Problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 169, pp. 391–405, 1999.

Magnetostatic Analysis

Products: Abaqus/Standard

Overview

Magnetostatic problems:

- solve the magnetostatic approximation of Maxwell's equations describing electromagnetic phenomena and compute the magnetic fields due to direct currents;
- involve only magnetic fields, which are assumed to be vary slowly in time such that electromagnetic coupling can be neglected;
- require the use of electromagnetic elements in the whole domain;
- require that magnetic permeability is specified in the whole domain;
- can be solved with nonlinear magnetic behavior; and
- can be solved using continuum elements in two- and three-dimensional space.

References:

- *Electromagnetic Analysis Procedures*
- *Magnetic Permeability*
- *Electromagnetic Loads*
- *Output to the Output Database*
- **MAGNETOSTATIC*
- **D EM POTENTIAL*
- **DECURRENT*
- **DSECURRENT*
- *UDECURRENT*
- *UDEMPOTENTIAL*
- *UDSECURRENT*

Magnetostatic Analysis

A direct current creates a static magnetic field in the space surrounding the current carrying region. For applications where the magnitude of the direct current can be assumed to be a constant or to vary slowly with time, coupling between magnetic and electric fields can be neglected. The magnetostatic approximation to Maxwell's equations involves the magnetic fields only. Magnetostatic analysis provides a solution for applications where the above assumptions are valid.

Electromagnetic elements must be used to model the response of all the regions in a magnetostatic analysis, including regions such as current carrying coils and the surrounding space. To obtain accurate solutions, the outer boundary of the space being modeled must be at least a few characteristic length scales away from the region of interest on all sides.

Electromagnetic elements use an element edge-based interpolation of the fields instead of the standard node-based interpolation. The user-defined nodes only define the geometry of the elements; and the degrees of freedom of

the element are not associated with these nodes, which has implications for applying boundary conditions (see *Boundary Conditions* below).

Governing Field Equations

The magnetic fields are governed by the magnetostatic approximation to Maxwell's equations describing electromagnetic phenomena.

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$. The solution procedure seeks a static magnetic response due to, for example, an impressed direct volume current density distribution, $\mathbf{J}$, in some regions of the model. The magnetostatic approximation to Maxwell's equations is given by

$$\nabla \times \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}\right) = \mathbf{J}$$

in terms of the field quantities, $\mathbf{A}$ and $\mathbf{J}$, and the magnetic permeability tensor, $\boldsymbol{\mu}$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A} \cdot \left(\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}\right)\, dV = \int_V \delta\mathbf{A} \cdot \mathbf{J}\, dV + \int_S \delta\mathbf{A} \cdot \mathbf{K}\ dS,$$

where $\delta\mathbf{A}$ represents the variation of the magnetic vector potential, and $\mathbf{K}$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential. In the following discussion, the governing equations are written for a linear medium.

Defining the Magnetic Behavior

The magnetic behavior of the electromagnetic medium can be linear or nonlinear. Linear magnetic behavior is characterized by a magnetic permeability tensor that is assumed to be independent of the magnetic field. It is defined through direct specification of the absolute magnetic permeability tensor, $\boldsymbol{\mu}$, which can be isotropic, orthotropic, or fully anisotropic (see *Magnetic Permeability*). The magnetic permeability can also depend on temperature and/or predefined field variables.

Nonlinear magnetic behavior is characterized by magnetic permeability that depends on the strength of the magnetic field. The nonlinear magnetic material model in Abaqus is suitable for ideally soft magnetic materials characterized by a monotonically increasing response in B–H space, where B and H refer to the strengths of the magnetic flux density vector and the magnetic field vector, respectively. Nonlinear magnetic behavior is defined through direct specification of one or more B–H curves that provide B as a function of H and, optionally, temperature and/or predefined field variables, in one or more directions. Nonlinear magnetic behavior can be isotropic, orthotropic, or transversely isotropic (which is a special case of the more general orthotropic behavior).

Magnetostatic Analysis

Magnetostatic analysis provides the magnetic flux density and the magnetic field at a given value of the impressed direct current.

Input File Usage: **MAGNETOSTATIC*

Ill-Conditioning in Magnetostatic Analyses

In magnetostatic analysis the stiffness matrix can be very ill-conditioned; i.e., it can have many singularities. Abaqus uses a special iterative solution technique to prevent the ill-conditioned matrix from negatively impacting the computed magnetic fields. The default implementation works well for many problems. However, there can be situations in which the default numerical scheme fails to converge. Abaqus provides a stabilization scheme to help mitigate the effects of the ill-conditioning. You can provide input to this stabilization algorithm by specifying the stabilization factor, which is assumed to be 1.0 by default if the stabilization scheme is used. Higher values of the stabilization factor lead to more stabilization, while lower values of the stabilization factor lead to less stabilization.

Input File Usage: **MAGNETOSTATIC*, STABILIZATION=*stabilization factor*

Initial Conditions

Initial values of temperature and/or predefined field variables can be specified. These values affect only temperature and/or field-variable-dependent material properties, if any. Initial conditions on magnetic fields cannot be specified in a magnetostatic analysis.

Boundary Conditions

Electromagnetic elements use an element edge-based interpolation of the fields. The degrees of freedom of the element are not associated with the user-defined nodes, which only define the geometry of the element. Consequently, the standard node-based method of specifying boundary conditions cannot be used with electromagnetic elements.

Boundary conditions in Abaqus typically refer to what are traditionally known as Dirichlet-type boundary conditions in the literature, where the values of the primary variable are known on the whole boundary or on a portion of the boundary. The alternative, Neumann-type boundary conditions, refer to situations where the values of the conjugate to the primary variable are known on portions of the boundary. In Abaqus, Neumann-type boundary conditions are represented as surface loads in the finite element formulation.

For electromagnetic boundary value problems, including magnetostatic problems, Dirichlet boundary conditions on an enclosing surface must be specified as $\mathbf{A} \times \mathbf{n}$, where $\mathbf{n}$ is the outward normal to the surface, as discussed in this section. Neumann boundary conditions must be specified as the surface current density vector, $\mathbf{K} = \mathbf{H} \times \mathbf{n}$, as discussed in *Loads* below.

In Abaqus, Dirichlet boundary conditions are specified as magnetic vector potential, $\mathbf{A}$, on (element-based) surfaces that represent symmetry planes and/or external boundaries in the model; Abaqus computes $\mathbf{A} \times \mathbf{n}$ for the representative surfaces. The model may span a domain that is up to 10 times some characteristic length scale for the problem. In such cases the magnetic fields are assumed to have decayed sufficiently in the far field, and the value of the magnetic vector potential can be set to zero in the far-field boundary. On the other hand, in applications such as one where a magnetic material is embedded in a uniform far-field magnetic field, it may be necessary to specify nonzero values of the magnetic vector potential on some portions of the external boundary. In this case an alternative method to model the same physical phenomena is to specify the corresponding unique value of surface current density, $\mathbf{K}$, on the far-field boundary (see *Loads* below). $\mathbf{K}$ can be computed based on known values of the far-field magnetic field.

In a magnetostatic analysis the boundary conditions are assumed to be either constant or varying slowly with time. The time variation can be specified using an amplitude definition (*Amplitude Curves*)

A surface without any prescribed boundary condition corresponds to a surface with zero surface currents or no loads.

When you prescribe the boundary condition on an element-based surface (see *Element-Based Surface Definition*), you must specify the surface name, the region type label (S), the boundary condition type label, an optional orientation name, the magnitude of the magnetic vector potential, and the direction vector for the magnetic vector potential. The optional orientation name defines the local coordinate system in which the components of the magnetic vector potential are defined. By default, the components are defined with respect to the global directions.

The specified vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

Nonuniform boundary conditions can be defined with user subroutine *UDEMPOTENTIAL*.

Input File Usage: Use the following option to define both the real (in-phase) and imaginary (out-of-phase) parts of the boundary condition on element-based surfaces:

**D EM POTENTIAL*
surface name, S, bc type label, orientation, magnitude, direction vector

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Loads

The following types of electromagnetic loads can be applied in a magnetostatic analysis (see *Prescribing Electromagnetic Loads for Eddy Current and/or Magnetostatic Analyses* for details):

- Element-based distributed volume current density vector, $\mathbf{J}(\mathbf{x})$
- Surface-based distributed surface current density vector, $\mathbf{K}(\mathbf{x})$

During the analysis the prescribed load can be varied using an amplitude definition (*Amplitude Curves*).

Predefined Fields

Predefined temperature and field variables can be specified in a magnetostatic analysis; however, user-defined fields that allow the value of field variables at a material point to be redefined via user subroutine *USDFLD* are not supported. These values affect only temperature and/or field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

The magnetic behavior (see *Magnetic Permeability*) must be defined everywhere in the model, either by specifying the absolute magnetic permeability tensor for linear magnetic behavior or by specifying the B–H curve-based response for nonlinear magnetic behavior. All other material properties, including electrical conductivity, are ignored in a magnetostatic analysis. The magnetic behavior can be functions of predefined temperature and/or field variables.

Permanent magnets (see *Magnetic Permeability*) can be included in magnetostatic analyses.

Elements

Electromagnetic elements must be used to model all regions in a magnetostatic analysis. Unlike conventional finite elements, which use node-based interpolation, these elements use edge-based interpolation with the tangential components of the magnetic vector potential along element edges serving as the primary degrees of freedom.

Electromagnetic elements are available in Abaqus/Standard in two dimensions (planar only) and three dimensions (see *Choosing the Appropriate Element for an Analysis Type*). The planar elements are formulated in terms of an in-plane magnetic vector potential, thereby the magnetic flux density and magnetic field vectors have only an out-of-plane component.

Output

Magnetostatic analysis provides output only to the output database (`.odb`) file (see *Output to the Output Database*). Output to the data (`.dat`) file and to the results (`.fil`) file is not available.

Element centroidal variables:

EMB Magnitude and components of the magnetic flux density vector, $\mathbf{B}$.

EMCDA Magnitude and components of the applied volume current density vector.

EMH Magnitude and components of the magnetic field vector, $\mathbf{H}$.

TEMP Temperature at the centroid of the element.

Whole element variables:

EVOL Element volume.

Input File Template

```
*HEADING
…
*MATERIAL, NAME=mat1
*MAGNETIC PERMEABILITY, NONLINEAR
Data lines to define magnetic permeability for linear magnetic behavior; no data required here for  nonlinear magnetic behavior
*NONLINEAR BH, DIR=direction
Data lines to define nonlinear B-H curve
**
*STEP
*MAGNETOSTATIC
Data line to define time incrementation
*D EM POTENTIAL
Data lines to define boundary conditions on magnetic vector potential
*DECURRENT
Data lines to define element-based distributed volume current density vector
*DSECURRENT
Data lines to define surface-based distributed surface current density vector
*OUTPUT, FIELD or HISTORY
Data lines to request element-based output
*END STEP
```

Electrochemical Analysis

Abaqus/Standard can solve several types of electrochemical problems that are intended for the analysis of battery electrochemistry.

In this section:

- *About Electrochemical Analysis Procedures*
- *Coupled Thermal-Electrochemical Analysis*
- *Fully Coupled Thermal-Electrochemical-Structural Analysis*
- *Fully Coupled Thermal-Electrochemical-Structural–Pore Pressure Analysis*
- *Modeling Solid Electrodes in Lithium Metal Batteries*
- *Modeling Aging in Batteries*
- *Modeling Thermal Runaway in Batteries*
- *Modeling Solid Electrolytes and Solid-State Batteries*
- *Parameter Table Type Reference*
- *Property Table Type Reference*

About Electrochemical Analysis Procedures

Overview

You can use several procedures for the analysis of battery electrochemistry to assess the battery cell's capacity due to different loading conditions, thermal effects, and mechanical effects.

In addition to the procedures available to model porous electrodes, you can also model solid electrodes in batteries, aging in lithium ion batteries, and solid electrolyte and solid state batteries.

Abaqus can solve the following types of electrochemical analyses:

Coupled thermal-electrochemical analysis

The coupled thermal-electrochemical procedure is intended for the analysis of battery electrochemistry applications that require solving simultaneously for temperature, electric potential in the solid electrode, electric potential in the electrolyte, concentration of ions in the electrolyte, and concentration in the solid particles used in the electrodes. In this procedure, the different fields are solved without any knowledge about the stress/deformation states. For more information, see *Coupled Thermal-Electrochemical Analysis*.

Fully coupled thermal-electrochemical-structural analysis

The fully coupled thermal-electrochemical-structural procedure is used to simultaneously solve for displacements, temperature, electric potential in the solid electrode, electric potential in the electrolyte, concentration of ions in the electrolyte, and concentration in the solid particles used in the electrodes. In this procedure, the thermal field and mechanical fields can affect each other. In addition, the concentration in the solid particles used in the electrodes can affect the mechanical fields through eigenstrains caused by particle swelling during the charge/discharge cycle in the battery. For more information, see *Fully Coupled Thermal-Electrochemical-Structural Analysis*.

Fully coupled thermal-electrochemical-structural–pore pressure analysis

The fully coupled thermal-electrochemical-structural–pore pressure procedure is used to simultaneously solve for displacements, pore fluid pressure that governs electrolyte flow, temperature, electric potential in the solid electrode, electric potential in the electrolyte, concentration of ions in the electrolyte, and concentration in the solid particles used in the electrodes. In this procedure, the thermal field and mechanical fields can affect each other. In addition, the concentration in the solid particles used in the electrodes can affect the mechanical fields through eigenstrains caused by particle swelling during the charge/discharge cycle in the battery. The fluid pressure and flow velocities in the electrolyte can be affected by the mechanical fields, gravity, and particle swelling during the charge/discharge cycle in the battery. For more information, see *Fully Coupled Thermal-Electrochemical-Structural–Pore Pressure Analysis*.

In addition to the procedures mentioned above to model porous electrodes, you can also model the following:

Solid electrodes in batteries

Solid particles are not modeled explicitly for a solid electrode. Instead, you define the electrochemistry on the interface between the solid electrode and the porous separator using surface-based loads or surface-based interactions. This approach results in a computationally efficient solution.

You can use any of the procedures listed above to model solid electrodes. For more information, see *Modeling Solid Electrodes in Lithium Metal Batteries*.

Aging in lithium ion batteries

A rechargeable lithium ion battery undergoes different degradation mechanisms that result in reduced capacity over time. The different aging mechanisms in rechargeable lithium ion batteries such as Solid Electrolyte Interface (SEI) layer growth, lithium plating, clogging of pores, and other phenomena can be modeled.

You can use any of the procedures listed above to model aging in lithium ion batteries. For more information, see *Modeling Aging in Batteries*.

Solid electrolyte and solid state batteries

Solid state batteries utilize a solid electrolyte, which also acts as a separator. The anode and cathode are solid electrodes with diffusion of species modeled at the cathode. Microscale simulations are not performed in a solid state battery. The interface between the solid electrolyte and the anode or cathode is modeled using surface-based loads or surface-based interactions.

You can use coupled thermal-electrochemical analysis and fully coupled thermal-electrochemical-structural analysis to model solid electrolyte and solid state batteries. For more information, see *Modeling Solid Electrolytes and Solid-State Batteries*.

Coupled Thermal-Electrochemical Analysis

Products: Abaqus/Standard

Overview

A coupled thermal-electrochemical analysis:

- is used for applications where the thermal, electrical, and ion concentration fields affect each other strongly;
- requires the use of coupled thermal-electrochemical elements;
- allows for transient or steady-state solutions for temperature and ion concentration and for steady-state solutions for solid and electrolyte electric potentials;
- can include the specification of a fraction of electrical and electrochemical energy that is released as heat; and
- can be linear or nonlinear.

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*
- *Fully coupled thermal-electrochemical procedures*
- **COUPLED THERMAL-ELECTROCHEMICAL*
- *Fully coupled thermal-electrochemical-structural analysis in sodium ion batteries*
- *Fully coupled thermal-electrochemical-structural analysis in lithium-ion batteries*

Typical Application

The primary example of a battery electrochemistry application is the charging and discharging of lithium-ion battery cells. During a charging cycle, the lithium ions are extracted from the active particles of the positive electrode (cathode). The ions move through the electrolyte by migration and diffusion from the positive electrode to the negative electrode across the separator. At the negative electrode (anode), the ions intercalate into the active particles. Heat is generated during the flow of electrons, ion migration, and the intercalation process. During discharge, the cycle is reversed.

Coupled Thermal-Electrochemical Analysis

Rechargeable lithium-ion batteries are widely used in a variety of applications, including portable electronic devices and electric vehicles. The performance of a battery highly depends on the effects of repeated charging and discharging cycles, which can cause the degradation of the battery capacity over time. The porous electrode theory (*Newman et al., 2004*) is commonly accepted as the leading method for modeling the charge-discharge behavior of lithium-ion cells. The method is based on a homogenized Newman-type approach that does not consider the details of the pore geometry. The porous electrode is based on the concurrent solution of a highly coupled multiphysics-multiscale formulation:

- At macroscale, the porous electrode is modeled as a homogenized medium consisting of the superposition of an active solid electrode particle phase and a liquid electrolyte solution phase, with known volume fractions.

- At microscale, a collection of spherical particles is assumed for which a (nonlinear) lithium-ion diffusion model is solved. The solid particles are connected by a conductive binder and together form the solid electrode phase.

You can use the coupled thermal-electrochemical procedure to model other battery electrochemical processes that use the governing equations described in the sections below. Although the discussion in the following sections is written in terms of lithium cells, the theory is general and can be used for other cell types.

A full stack lithium-ion battery cell consists of an anode collector, porous anode, porous separator, porous cathode, and cathode collector. The porous electrode can be described as a material with a solid skeleton and a distribution of interconnected pores that are filled with the electrolyte. The solid skeleton consists of a large collection of active spherical particles that are in contact with the electrolyte. The spherical particles are electrically conductive and are in electrical contact among themselves, but they do not allow for interparticle diffusion. The spherical particles are also chemically active and allow for intercalation and deintercalation of lithium ions.

Figure 1 shows a typical jelly roll configuration of a cylindrical battery. The porous parts of the battery are immersed in an electrolyte bath that facilitates the movement of ions from the cathode to the anode during the charging cycle (*Figure 2*). The porous electrodes are connected to nonporous collectors that are connected to the external terminal of the battery. The external terminals of the battery are connected to a power source during the charging cycle. The electrons move in the solid phase, while the ions move in the liquid phase.

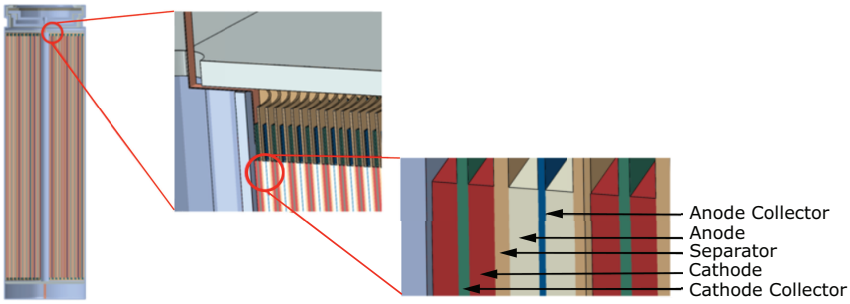


Figure 1: Schematic diagram of a full stack lithium-ion battery cell.

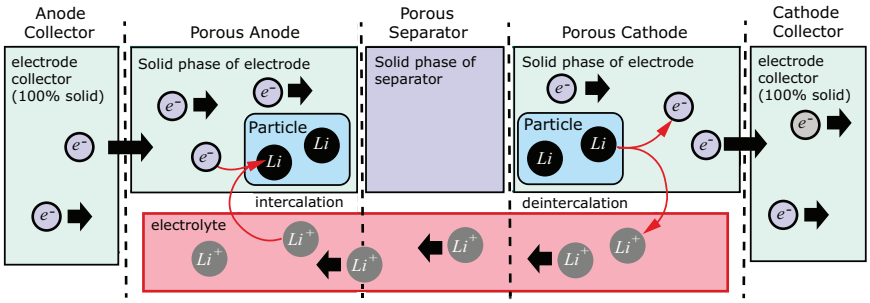


Figure 2: Schematic diagram of the charging process in a full stack lithium-ion battery cell.

At the cathode solid-liquid interface, lithium ions can move out of the solid electrode particles and enter the electrolyte through a deintercalation (or extraction) process that can be written as:

$$\mathbf{Li} - \boldsymbol{\Theta} \leftrightarrow \mathbf{Li}^{+} + e^{-} + \boldsymbol{\Theta}.$$

Here, $\mathbf{Li} - \boldsymbol{\Theta}$ represents lithium in the solid electrode, which through the deintercalation reaction, transforms into a lithium ion in the electrolyte, $\mathbf{Li}^{+}$, plus an electron, e^{-}, and leaves behind a vacancy (or intercalation site) in the solid, $\boldsymbol{\Theta}$. The reverse reaction corresponds to the intercalation of lithium into the solid electrode.

During the charging cycle, deintercalated lithium ions at the cathode solid-liquid interface move in the electrolyte through the separator and reach the anode. At the anode, the lithium ions are intercalated (inserted) into the solid electrode particles. The separator is porous and allows ions to pass through; however, it is an electrically insulated material. Therefore, the electrons cannot pass through, which avoids the possibility of a short circuit in the cell.

The governing equations used to model lithium-ion cells span both multiple scales and multiple physics and involve solving a fully coupled thermal and electrochemistry problem. The primary solution variables that are solved for at the macroscale (or continuum scale) are the electric potential in the solid phase (Φ_s), temperature (θ), electric potential in the electrolyte (Φ_e), and the lithium ion concentration (C_e). At microscale, the model solves for the concentration of lithium (C_s) in the particle.

Governing Equations

The porous electrode theory consists of five main parts:

- Intercalation at the solid-liquid interface.
- Conduction of electrons in the solid phase.
- Diffusion and migration of lithium ions in the liquid phase.
- Diffusion of lithium in the microscale particle.
- Coupling of the continuum scale and microscale.

An optional sixth contribution is associated with Joule heating generation.

Intercalation at the Solid-Liquid Interface

The intercalation process at the solid-liquid interface is governed by electrochemical kinetics and is defined by the Butler-Volmer equation. The Butler-Volmer equation describes the relationship between the electric current density through the solid electrode-electrolyte interface, I_{BV}, and the voltage difference between the electrode and electrolyte according to:

$$I_{\mathrm{BV}} = I_0\left\{\exp\left[\frac{\alpha_a(zF\eta - \sigma_m\Omega)}{R(\theta-\theta^z)}\right] - \exp\left[\frac{-\alpha_c(zF\eta - \sigma_m\Omega)}{R(\theta-\theta^z)}\right]\right\},$$

where I_0 and η are the exchange current density and overpotential, respectively; σ_m is the mean stress or hydrostatic pressure; and Ω represents the partial molar volume of lithium in the electrode material. The term $\sigma_m\Omega$ captures the effects of mechanical stress on the electrical response of the electrode material. The overpotential, η, is defined as:

$$\eta = \Phi_s - \Phi_e - E_{\mathrm{ref}}\left(\widehat{C}_{s,P}^{\mathrm{surf}}, \theta\right) - R_{SEI}I_{BV},$$

where R_{SEI} is the solid electrolye interface resistance.

The exchange current density, I_0, can be written as:

$$I_0 = Fk_c^{\alpha_a}k_a^{\alpha_c}C_{s,P}^{\max}\left(1-\widehat{C}_{s,P}^{\mathrm{surf}}\right)^{\alpha_a}\left(\widehat{C}_{s,P}^{\mathrm{surf}}\right)^{\alpha_c}\left(\frac{C_e}{C_e^{\mathrm{ref}}}\right)^{\alpha_a}.$$

In the above equations,

F
: is Faraday's constant;

R
: is the gas constant;

z
: is the charge number of the lithium ion battery;

θ
: is temperature;

θ^z
: is the absolute zero temperature,

k_c, k_a
: are the cathodic and anodic rate constants, respectively;

α_c, α_a
: are the cathodic and anodic transfer coefficients, respectively;

$C_s^{\max}$
: is the theoretical maximum lithium capacity;

$\widehat{C}_{s,P}^{\text{surf}}$
: is the normalized surface lithium concentration in the microscale particle;

C_e^{ref}
: is the reference value of lithium ion concentration in the electrolyte; and

E_{ref}
: is the open circuit potential (OCP) as a function of $\widehat{C}_{s,P}^{\text{surf}}$ and θ.

The normalized concentration, $\widehat{C}_{s,P}$, is defined as $\widehat{C}_{s,P} = \frac{C_{s,P}}{C_{s,P}^{\max}}$, where $C_{s,P}$ is the lithium concentration in the particle. The normalized surface concentration, $\widehat{C}_{s,P}^{\text{surf}}$, is defined as $\widehat{C}_{s,P}^{\text{surf}} = \frac{C_{s,P}^{\text{surf}}}{C_{s,P}^{\max}}$, where $C_{s,P}^{\text{surf}}$ is the lithium concentration on the surface of the particle.

Conduction of Electrons in the Solid Phase

The equilibrium equation for the current density, $\mathbf{I}_s$, in the solid phase is given by:

$$\nabla \cdot \mathbf{I}_s = -a_s I_{\text{BV}},$$

where

$$\mathbf{I}_s = -\mathbf{K}_s^{\text{eff}} \cdot \nabla \Phi_s.$$

$\mathbf{K}_s^{\text{eff}}$ is the effective electrical conductivity of the solid phase computed using the Bruggeman relationship for tortuosity ($\tau_s = \varepsilon_s^{-\alpha}$). Tortuosity represents an intrinsic property of a porous material characterizing the ratio of the actual flow path length to the straight distance between the ends.

For isotropic tortuosity:

$$\mathbf{K}_s^{\text{eff}} = \frac{\varepsilon_s}{\tau_s} K_s \mathbf{I} = \varepsilon_s^{(1+\alpha)} K_s \mathbf{I}.$$

For anisotropic tortuosity, the above expression can be written as:

$$\mathbf{K}_s^{\text{eff}} = \begin{bmatrix} \varepsilon_s^{(1+\alpha_x)} & 0 & 0 \\ 0 & \varepsilon_s^{(1+\alpha_y)} & 0 \\ 0 & 0 & \varepsilon_s^{(1+\alpha_z)} \end{bmatrix} K_s,$$

where

K_s
: is the electric conductivity of the bulk solid material (which can be a function of normalized concentration);

τ_s
: is the tortuosity of the solid porous electrode;

ε_s
: is the volume fraction of the solid phase;

$\mathbf{I}$
: is the unit tensor;

α
: is the Bruggeman exponent for isotropic tortuosity;

$\alpha_x, \alpha_y, \alpha_z$
: are the Bruggeman exponents for anisotropic tortuosity; and

a_s
: is the wetted particle surface area per unit volume, computed by default as $a_s = \frac{3\varepsilon_s}{R_p}$, where R_p is the outer radius of the spherical particle associated with the microscale model.

Diffusion and Migration of Lithium Ions in the Liquid Phase

The current density in the electrolyte, $\mathbf{I}_e$, and the lithium ion concentration, C_e, in the liquid phase are solved for using the following governing equations:

$$\nabla \cdot \mathbf{I}_e = a_s I_{\text{BV}},$$

where

$$\mathbf{I}_e = -\mathbf{K}_e^{\text{eff}} \cdot \left[\nabla \Phi_e - \frac{2\,R(\theta - \theta^z)}{F} \left(1 + \frac{\mathrm{d} \ln f_{\pm}}{\mathrm{d} \ln C_e} \right) (1 - t_+) \nabla \ln C_e \right],$$

$$\frac{\partial(\varepsilon_e C_e)}{\partial t} = -\nabla \cdot \mathbf{J}_e + \frac{a_s I_{\mathrm{BV}}}{F},$$
$$\mathbf{J}_e = -\mathbf{D}_e^{\mathrm{eff}} \cdot \nabla C_e + \frac{t_+}{F}\mathbf{I}_e.$$

In the above equations,

$\mathbf{K}_e^{\mathrm{eff}}$
: is the effective ionic conductivity in the electrolyte phase,

$\mathbf{D}_e^{\mathrm{eff}}$
: is the effective ionic diffusivity in the electrolyte phase,

t_+
: is the transference number that defines the fraction of the total electrical current that is carried by lithium ions in the electrolyte, and

$\left(\frac{\mathrm{d\ ln\ f_\pm}}{\mathrm{d\ ln\ C_e}}\right)$
: is the molar activity coefficient that accounts for deviations from the ideal behavior in a mixture of chemical substances.

$\mathbf{K}_e^{\mathrm{eff}}$ and $\mathbf{D}_e^{\mathrm{eff}}$ are computed using the Bruggeman relationship for tortuosity.

For isotropic tortuosity:

$$\mathbf{K}_e^{\mathrm{eff}} = \frac{\varepsilon_e}{\tau_e} K_e \mathbf{I} = \varepsilon_e^{(1+\alpha)} K_e \mathbf{I},$$
$$\mathbf{D}_e^{\mathrm{eff}} = \frac{\varepsilon_e}{\tau_e} D_e \mathbf{I} = \varepsilon_e^{(1+\alpha)} D_e \mathbf{I}.$$

For anisotropic tortuosity, the above expression can be written as:

$$\mathbf{K}_e^{\mathrm{eff}} = \begin{bmatrix} \varepsilon_e^{(1+\alpha_x)} & 0 & 0 \\ 0 & \varepsilon_e^{(1+\alpha_y)} & 0 \\ 0 & 0 & \varepsilon_e^{(1+\alpha_z)} \end{bmatrix} K_e,$$
$$\mathbf{D}_e^{\mathrm{eff}} = \begin{bmatrix} \varepsilon_e^{(1+\alpha_x)} & 0 & 0 \\ 0 & \varepsilon_e^{(1+\alpha_y)} & 0 \\ 0 & 0 & \varepsilon_e^{(1+\alpha_z)} \end{bmatrix} D_e,$$

where

K_e, D_e
: are the ionic conductivity and ionic diffusivity of the electrolyte, respectively;

$\mathbf{I}$
: is the unit tensor;

τ_e
: is the tortuosity of the liquid (electrolyte) phase;

ε_e
: is the volume fraction of the liquid phase;

α
: is the Bruggeman exponent for isotropic tortuosity; and

$\alpha_x, \alpha_y, \alpha_z$
: are the Bruggeman exponents for anisotropic tortuosity.

Diffusion of Lithium in the Microscale Particle

The active material at the electrodes is assumed to consist of spherical microparticles. The particles are assumed to have electrical contact between themselves but no interparticle diffusion. It is assumed that the microparticle material is a good electronic conductor (transport number=1) and has spherical symmetry. The conservation of lithium inside the particle is governed by Fick's second law of diffusion, written in spherical coordinates:

$$\frac{\partial C_{s,P}}{\partial t} = -\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 j_s\right),$$

$$j_s = D_s\left(C_{s,P}\right)\frac{\partial C_{s,P}}{\partial r},$$

where

r
: is the radial coordinate of the spherical particle,

$C_{s,P}$
: is the lithium concentration in the particle, and

$D_s(C_{s,P})$
: is the concentration-dependent diffusion constant of the solid material.

The time evolution of lithium concentration in the particle is determined by diffusion and the intercalation current density, I_{BV}, from the electrochemical model in the continuum scale. The microscale particle is modeled as a sphere over which a one-dimensional finite element mesh in spherical coordinates is created internally by Abaqus, and the diffusion problem is solved. You can specify the discretization for the internal mesh and the type of meshing over the microscale particle (see *Particle Layer Mesh*).

Coupling of the Continuum Scale and Microscale

The coupling of the microscale with the continuum scale happens through the Butler-Volmer current density, I_{BV}, from the electrochemical model:

$$j_s = \frac{I_{\mathrm{BV}}}{F}, \text{ at } r = R_P.$$

Heat Transfer and Joule Heating

The total heat generation in a battery is attributed to the contribution of five different sources: flow of current at the solid-liquid interface, flow of current in the solid phase, flow of current in the liquid phase, flow of ions, and entropy generation. The amount of energy converted into heat from each of the source terms can be scaled using the conversion factors, β_α (α=1–5):

$$Q_{\text{tot}} = \beta_1 Q_1 + \beta_2 Q_2 + \beta_3 Q_3 + \beta_4 Q_4 + \beta_5 Q_5,$$

where

$Q_1 = a_s I_{\text{BV}} \eta$
is the ohmic loss at the solid-liquid interface;

$Q_2 = a_s I_{\text{BV}} \left(\theta - \theta^z\right) \frac{\partial \text{U}}{\partial \theta}$
is the entropy generation, where $\frac{\partial \text{U}}{\partial \theta}$ is the derivative of the open circuit potential with respect to temperature that you can specify in tabular form;

$Q_3 = \mathbf{K}_s^{\text{eff}} \nabla \Phi_s \cdot \nabla \Phi_s$
is the ohmic loss in the electrode;

$Q_4 = \mathbf{K}_e^{\text{eff}} \nabla \Phi_e \cdot \nabla \Phi_e$
is the ohmic loss in the electrolyte; and

$Q_5 = -\frac{2R(\theta - \theta^z)}{F} \mathbf{K}_e^{\text{eff}} \left(1 + \frac{\text{d} \ln f_\pm}{\text{d} \ln C_e}\right) (1 - t_+) \nabla \ln C_e \cdot \nabla \Phi_e$
is the ohmic loss due to ion diffusion.

Fully Coupled Solution Scheme

The coupled thermal-electrochemical analysis in Abaqus uses an exact implementation of Newton's method, leading to an unsymmetric Jacobian matrix in the form:

$$\begin{bmatrix} K_{\varphi_s \varphi_s} & K_{\varphi_s \theta} & K_{\varphi_s \varphi_e} & K_{\varphi_s C_e} \\ K_{\theta \varphi_s} & K_{\theta\theta} & K_{\theta \varphi_e} & K_{\theta C_e} \\ K_{\varphi_e \varphi_s} & K_{\varphi_e \theta} & K_{\varphi_e \varphi_e} & K_{\varphi_e C_e} \\ K_{C_e \varphi_s} & K_{C_e \theta} & K_{C_e \varphi_e} & K_{C_e C_e} \end{bmatrix} \begin{Bmatrix} \Delta \varphi_s \\ \Delta \theta \\ \Delta \varphi_e \\ \Delta \text{C}_e \end{Bmatrix} = \begin{Bmatrix} R_{\varphi_s} \\ R_\theta \\ R_{\varphi_e} \\ R_{C_e} \end{Bmatrix}.$$

Steady-State Analysis

Steady-state analysis provides the steady-state solution by neglecting the transient terms in the continuum scale equations. It can be used to achieve a balanced initial state or to assess conditions in the cell after a long storage period.

In the thermal equation, the internal energy term in the governing heat transfer equation is omitted. Similarly, the transient term is omitted in the diffusion equations for the lithium ion concentration in the electrolyte. Electrical transient effects are not included in the equations because they are very rapid compared to the

characteristic times of thermal and diffusion effects. A steady-state analysis has no effect on the microscale solution; the transient terms are always considered in the solution of the lithium concentration in the solid particle.

Input File Usage: **COUPLED THERMAL-ELECTROCHEMICAL*, STEADY STATE

Transient Analysis

In a transient analysis, the transient effects in the heat transfer and diffusion equations are included in the solution. Electrical transient effects are always omitted because they are very rapid compared to the characteristic times of thermal and mass diffusion effects.

Input File Usage: **COUPLED THERMAL-ELECTROCHEMICAL*

Spurious Oscillations due to Small Time Increments

By default, Abaqus/Standard uses nodal integration of the heat capacity term for first-order elements in a transient analysis, resulting in a lumped treatment for this term. This treatment eliminates nonphysical oscillations; however, it can lead to locally inaccurate solutions, especially in terms of the heat flux for small time increments. If smaller time increments are required, you should use a finer mesh in regions where the temperature changes occur. For more information about time integration involving the temperature degree of freedom, see *Spurious Oscillations due to Small Time Increments* in *Uncoupled Heat Transfer Analysis*.

Controlling the Analysis Based on Solution Variables

A battery analysis typically consists of four (or more) steps in the following order:

- A charging step, in which the battery is subjected to a constant charging current. The voltage is monitored, and the step is terminated when the voltage increases to a specified cut-off value.
- A constant voltage step, in which the battery is subjected to a constant prescribed voltage. The reaction current is monitored, and the step is terminated when the reaction current drops below a specified threshold value.
- A resting step, in which the battery is not subjected to any prescribed current or voltage.
- A discharge step, in which a constant current is drawn from the battery. The voltage is monitored, and the step is terminated when the voltage drops to a specified cut-off value.

You can use the step control mechanism (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) together with sensors (see *Defining Sensors*) to model the workflow described above. In particular, you can associate the history output of the voltage or the current at specific locations in the model with sensor definitions and terminate a step when the sensor reaches a predefined cut-off value or threshold.

Initial Conditions

By default, the initial values of electric potential in the solid, temperature, electric potential in the electrolyte, and ion concentration of all nodes are set to zero. You can specify nonzero initial values for the primary solution variables (see *Initial Conditions*).

The typical set of initial conditions includes uniform but different values for the two electrodes. The initial lithium concentration in the particles and solid and fluid electric potentials in the electrodes are chosen such that the overpotential in the electrodes is zero. The ion concentration in the electrolyte is typically assumed to be uniform in the cell.

The initial condition for the microscale spherical particle is:

$$C_{s,P}(r) = C_{s,P}^{\text{initial}}, \text{ at } t = 0.$$

You can specify the initial concentration of the spherical particle using the parameter table *"ABQ_EChemPET_Electrode_Particles"*.

Boundary Conditions

You can prescribe the following boundary conditions:

- Electric potential in the solid, $\varphi_s = \varphi_s(\mathbf{x}, t)$ (degree of freedom 9).
- Electric potential in the electrolyte, $\varphi_e = \varphi_e(\mathbf{x}, t)$ (degree of freedom 32).
- Temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11).
- Ion concentration in the electrolyte, $C_e = C_e(\mathbf{x}, t)$ (degree of freedom 33) at the nodes.

You can specify boundary conditions as functions of time by referring to amplitude curves.

A boundary without any prescribed boundary conditions corresponds to an insulated (zero flux) surface.

The typical boundary condition consists only of grounding (setting to zero) the solid electric potential at the anode. Thermal boundary conditions vary.

At the microscale, the boundary condition for the spherical particle is:

$$\frac{\partial C_{s,P}}{\partial r} = 0, \text{ at } r = 0.$$

Abaqus applies the microscale boundary conditions automatically.

Loads

You can apply thermal, electrical, and electrochemical loads in a coupled thermal-electrochemical analysis.

You can prescribe the following types of thermal loads (as described in *Thermal Loads*):

- Concentrated heat flux.
- Body flux and distributed surface flux.
- Convective film and radiation conditions.

You can prescribe the following types of electrical loads on the solid (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of electrical loads on the electrolyte (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of ion concentration loads (as described in *Thermal Loads*):

- Concentrated flux.
- Distributed body flux.

The typical loads include specification of a solid electric flux (current) at the cathode. Thermal boundary conditions vary but typically include convective film on the exterior surfaces. Customarily, no loads are applied on the concentrations and electrolyte potential.

Predefined Fields

Predefined temperature fields are not allowed in coupled thermal-electrochemical analyses. You can use boundary conditions to specify temperatures. You can specify other predefined field variables in a coupled thermal-electrochemical analysis. These values affect only field variable–dependent material properties.

Material Options

The thermal and electrical properties for both the solid and the electrolyte are active in a coupled thermal-electrochemical analysis. All mechanical behavior material models (such as elasticity and plasticity) are ignored in a coupled thermal-electrochemical analysis. The electrochemistry framework requires that the material definition contain the complete specification of properties required for the porous electrode theory, as described below and in the sections that follow. In addition, the material name must begin with "ABQ_EChemPET_" to enable the coupled micro-macro solution at the different electrodes. Special-purpose parameter and property tables of type names starting with "ABQ_EChemPET_" are required in these material definitions (see *Parameter Table Type Reference* and *Property Table Type Reference*).

Input File Usage:
**MATERIAL*, NAME=ABQ_EChemPET_*name*
**POROUS ELECTRODE THEORY*

Thermal Material Properties

You must define thermal conductivity for the heat transfer portion of the analysis. In addition, you must define the specific heat for transient problems. Thermal expansion coefficients are not meaningful in a coupled thermal-electrochemical analysis because the deformation of the structure is not considered. You can specify internal heat generation.

Input File Usage:
**MATERIAL*
**CONDUCTIVITY*
**DENSITY*
**SPECIFIC HEAT*

Electrical Properties for the Solid Electrode Material

You must define the electric conductivity for the solid electrode portion of the analysis. The electrical conductivity is defined as a function of the average normalized concentration of the particle, $\widehat{C}_s$ (and, optionally, of temperature and user-defined field variables).

Input File Usage: Use the following options to define the electric conductivity of the solid phase:

**MATERIAL*
**PROPERTY TABLE*, TYPE=*"ABQ_EChemPET_Electrode_ElecCond_Tabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_ElecCond_Tabular"

K_s, $\widehat{C}_s$, θ, **FV**

Ionic Properties for the Electrolyte Material

You define the electrolyte name and the ion charge number (z). The charge number for the lithium ion is 1.0.

A label with the electrolyte name is used to identify all subsequent definitions required to specify the electrical and ion diffusion properties of the electrolyte.

You define the ionic conductivity, diffusion coefficient, molar activity coefficient, and the transference (migration). The ionic conductivity and diffusion coefficients are functions of the lithium ion concentration and the electrolyte volume fraction in the electrode. The molar activity coefficient and the transference are functions only of the lithium ion concentration. In addition, all the electrolyte properties can depend on temperature and field variables. You can define Arrhenius temperature dependency for the ionic conductivity and diffusivity. When you define temperature dependence using a property table, the Arrhenius definition is ignored. For more details about the Arrhenius dependency, see *Arrhenius Temperature Dependency*.

Input File Usage: Use the following options to define the electrolyte name and charge number:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Electrolyte"*,
LABEL="ABQ_EChemPET_Electrolyte"
"Electrolyte name", z

Use the following options to define the ionic conductivity of the electrolyte:

**MATERIAL*
**PROPERTY TABLE*, TYPE=*"ABQ_EChemPET_Electrolyte_ElecCond_Tabular"*,

LABEL="ABQ_EChemPET_<*Electrolyte name*>_ElecCond_Tabular"
K_e, C_e, ε_e, θ, **FV**
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Arrhenius"*,
LABEL="ABQ_EChemPET_<*Electrolyte name*>_CondArrhenius"
E_a, θ_{ref}

Use the following options to define the diffusion coefficient of the electrolyte:

**MATERIAL*
**PROPERTY TABLE*, TYPE=*"ABQ_EChemPET_Electrolyte_DiffTabular"*,
LABEL="ABQ_EChemPET_<*Electrolyte name*>_DiffTabular"
D_e, C_e, ε_e, θ, **FV**
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Arrhenius"*,
LABEL="ABQ_EChemPET_<*Electrolyte name*>_DiffArrhenius"
E_a, θ_{ref}

Use the following options to define the molar activity coefficient of the electrolyte:

**MATERIAL*
**PROPERTY TABLE*,
TYPE=*"ABQ_EChemPET_Electrolyte_MolarActivityCoeff_fPM"*,
LABEL="ABQ_EChemPET_<*Electrolyte name*>_MolarActivityCoeff_fPM"

$\frac{\mathrm{d}\ln \mathrm{f}_{\pm}}{\mathrm{d}\ln \mathrm{C}_e}$, C_e, θ, **FV**

Use the following options to define the transference for the electrolyte:

**MATERIAL*
**PROPERTY TABLE*, TYPE=*"ABQ_EChemPET_Electrolyte_Transference"*,
LABEL="ABQ_EChemPET_<*Electrolyte name*>_Transference"
t_+, C_e, θ, **FV**

Defining Material Properties for the Porous Electrode and Microscale Particle

The following sections describe the material properties used to define the multiscale nature of the electrodes and separator progressing from the macroscale electrode level to the microscale particle level.

Electrode Definition

You must define various properties for the electrode at the macroscale, including a unique region name; a region identifier characterizing the electrode as either an anode, cathode, or separator; the volume fractions of the various phases in the electrode; and the tortuosity factors in the three material directions.

You can specify a utilization fraction to account for inaccessible regions in the electrode (see *Ecker, 2015).*

When lithium ions intercalate into a particle, the particle can swell resulting in a convection of the electrolyte away from that region. If this effect is significant and of interest to you, you can specify a factor of one for the convection coefficient. By default, the value for the convection coefficient is zero.

When particle swelling effects are included, the binder volume fraction, ε_b, and solid volume fraction, ε_s, are used to compute and update the electrolyte volume fraction, ε_e.

Input File Usage: Use the following options to define the electrode name; region type (anode, cathode, or separator); and other electrode properties:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Electrode_Definition"*,
LABEL="ABQ_EChemPET_Electrode_Definition"
"Electrode name", *"Region Type"*, ε_s, ε_e, ε_b, $\varepsilon_{\text{inactive}}$, "*Utilization Fraction*", , α_z, α_x, α_y, $\text{Conv}_{\text{coef}}$

Distinguishing Fluid and Solid Phase Tortuosities

By default, the Bruggeman exponents defined during the electrode definition are used to determine the tortuosities for both the solid and fluid phases of the electrode. However, you can define different Bruggeman exponents to be used to determine the effective electric conductivity in the solid phase and the effective ionic conductivity and ionic diffusivity of the electrolyte phase.

Input File Usage: Use the following options to define the Bruggeman exponents for the effective electric conductivity in the solid phase:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*ABQ_EChemPET_Electrode_SolidVF_BrugExp*,
LABEL="ABQ_EChemPET_<*Electrode name*>_SolidVF_BrugExp"
α_z^s, α_x^s, α_y^s

Use the following options to define the Bruggeman exponents for the effective ionic conductivity and ionic diffusivity of the electrolyte or liquid phase:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*ABQ_EChemPET_Electrode_FluidVF_BrugExp*,
LABEL="ABQ_EChemPET_<*Electrode name*>_FluidVF_BrugExp"
α_z^e, α_x^e, α_y^e

Particle Definition

For each electrode you must define particle-specific properties such as the particle radius, the initial concentration of the particle, and the contribution factor of the particle when multiple particles are present. You can define multiple particle types within the same electrode.

Input File Usage: Use the following options to define the name and properties of the particle inside an electrode:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Electrode_Particles"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_Particles"

"Particle name", R_p, $\varepsilon_{\text{sParticle}}$, $C_{s,P,\text{Init}}$, *"Formulation Type"*

Repeat the data line as often as necessary to define a list of particles within the same electrode.

Use the following option to define a distribution (see *Distribution Definition*) of particle radius that changes from one element to another:

**MATERIAL*
**PARAMETER TABLE*,
TYPE="*ABQ_EChemPET_Electrode_Particle_radiusDistribution*",
LABEL="ABQ_EChemPET_<*Electrode name*>_Particle_radiusDistribution"
"Distribution Table Name"

Particle Layer Definition

Each particle can have one layer or multiple concentric layers. All layers within a particle must have the same material. For each layer, you must define a name for the layer and a weight percentage per layer.

Input File Usage: Use the following options to define one or more layers:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Electrode_Particle_Layers"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_*Layers*"
"Layer name", *"Weight Fraction of Layer"*

Repeat the data line as often as necessary to define additional layers.

Use the following options to define the open circuit potential for a layer in tabular format:

**MATERIAL*
**PROPERTY TABLE*,
TYPE=*"ABQ_EChemPET_Electrode_Particle_Layer_OCPTabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_OCPTabular"

E_{ref}, $\widehat{C}_{s,P}^{\text{surf}}$, θ, FV

Use the following options to define the open circuit potential for a layer in spline format:

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemPET_CubicSplineC2",
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_OCPCubicSpline"

X_i, X_{i+1}, C_3, C_2, C_1, C_0

Repeat the data line as often as necessary to cover the full range of the independent variable, X.

Particle Layer Mesh

The microscale particle is modeled as a sphere over which a one-dimensional finite element mesh in spherical coordinates is created and the diffusion problem is solved. You must specify the discretization ($N_{\text{elemPerLayer}}$) to use on each layer of the microscale particle. The total number of elements in a particle can be between 1 and 100. Typically, a mesh of 25 elements on a single particle gives good results. The meshing can be uniform or biased toward the outside of the sphere using a quadratic or equal volume approach.

Input File Usage: Use the following options to define the mesh discretization within a particle layer:

*MATERIAL
*PARAMETER TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_Layer_Discretization",
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_Discretization"

$N_{\text{elemPerLayer}}$, *"Mesh Type"*

Particle Layer Diffusion

You must define the theoretical maximum lithium capacity, $C_{s,P}^{\max}$, of each layer material in the particle and the diffusion coefficient to use in the solution of the microscale lithium diffusion equations. Several formats are available to define the diffusion coefficient as a function of the normalized concentration, $\hat{C}_{s,P}$: tabular format, logarithmic tabular format, and spline format.

- For a tabular diffusion model, the diffusion coefficient is defined as a piecewise linear function of the normalized particle layer concentration.
- For the logarithmic definition of the layer diffusion, you specify the base 10 logarithm of the diffusion coefficient as a tabular function of the normalized particle layer concentration.
- The spline input form for the diffusion curve requires the diffusion curve as a function of the normalized concentration to be split into equal intervals along the *x*-axis. Each interval is then curve fit with a cubic spline equation. The parameter table has entries ordered as the start and end of the interval range and the four polynomial coefficients that correspond to the curve fit within that range.

In addition, the diffusion coefficient can be a function of temperature. For the spline format, only the Arrhenius form of temperature dependency is supported.

Input File Usage: Use the following options to define the theoretical maximum lithium capacity and the diffusion model type for a given particle layer:

*MATERIAL
*PARAMETER TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_Layer_Diffusion",
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer*

name>_Diffusion"

$C_{s,P}^{\max}$, *"Diffusion Data Type"*, *"Chemical Diffusion Type"*

Use the following options to define the diffusion coefficient using a tabular format:

**MATERIAL*
**PROPERTY TABLE*,
TYPE=*"ABQ_EChemPET_Electrode_Particle_Layer_DsTabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_DsTabular"

D_s, $\widehat{C}_{s,P}$

Use the following options to define the diffusion coefficient using a logarithmic tabular format:

**MATERIAL*
**PROPERTY TABLE*, TYPE=*"ABQ_EChemPET_LogScale_Tabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_DsLogTabular"

$\text{Log}_{10}\, D_s$, $\widehat{C}_{s,P}$

Use the following options to define the diffusion coefficient for a layer in spline format:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_CubicSplineC2"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_DsCubicSpline"

X_i, X_{i+1}, C_3, C_2, C_1, C_0

Repeat the data line as often as necessary to cover the full range of the independent variable, X.

Use the following options to define the coefficients of the Arrhenius model:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Arrhenius"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_DiffArrhenius"

E_a, θ_{ref}

Particle Swelling

The intercalation/deintercalation process of lithium atoms can lead to significant changes in the volume of the spherical particles, which increases with the average lithium concentration within the particle. Particle swelling also results in the local displacement of the electrolyte (convection effects) to accommodate the new particle volume. While elastic deformation is not modeled explicitly by the thermal-electrochemical procedure, the effects of particle swelling on the electrochemical process are accounted for by considering its influence on the effective volume fractions of the solid and liquid phases, as well as on the wetted particle surface area per unit volume (a_s).

The volume change in the particle due to a change in average concentration is characterized by a swelling coefficient, Ω, as:

$$\frac{\Delta V_p}{V_p^0} = \Omega \, \Delta \, c_{s,P}^{\text{avg}}.$$

The solid volume fraction is defined as

$$\varepsilon_s = \frac{n_p V_p^0}{V_{\text{macro}}},$$

where n_p is the number of particles in the macroscopic volume, and V_p^0 is the particle initial volume.

Assuming that the local particle volume change is accommodated by displacing away the electrolyte, with no overall macroscopic volume change, then

$$\Delta\varepsilon_s = \frac{n_p \Delta V_p}{V_{\text{macro}}} = \frac{n_p V_p^0 \Omega \Delta c_{s,P}^{\text{avg}}}{V_{\text{macro}}} = \varepsilon_s \Omega \Delta c_{s,P}^{\text{avg}},$$

and, for the electrolyte

$$\Delta\varepsilon_e = -\Delta\varepsilon_s = -\varepsilon_s \Omega \Delta c_{s,P}^{\text{avg}} = -\left(1 - \varepsilon_e\right) \Omega \Delta c_{s,P}^{\text{avg}}.$$

Similarly, the active surface area per volume, a_s, is also affected by swelling with

$$\Delta a_s = \frac{2}{3} \, a_s \Omega \Delta c_{s,P}^{\text{avg}}.$$

The electrochemical governing equations are modified accordingly to incorporate the effects of particle swelling on ε_s, ε_e, and a_s.

You can include the effects of particle swelling in the simulation by specifying (in tabular form) volume ratios at different values of the normalized concentration.

Input File Usage: Use the following options to define the volume ratio of the current volume at the specified concentration, V_p, to the reference volume, V_p^0. This ratio is used to determine the volume fraction change during lithiation or delithiation as a function of normalized particle concentration:

**MATERIAL*
**PROPERTY TABLE*,
TYPE=*"ABQ_EChemPET_Electrode_Particle_Layer_SwellingTabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_<*Layer name*>_SwellingTabular"

$$\frac{V_p}{V_p^0}, \widehat{C}_{s,P}$$

Butler-Volmer Definition

You must define the constants required to compute the current density using the Butler-Volmer equation. You can choose whether the particle surface area per unit volume, a_s, is computed within Abaqus or specified directly as a single value. By default, Abaqus computes the current exchange density, I_0. You can specify $\overline{I_0}$ in tabular form as a function of the normalized particle concentration and temperature. $\overline{I_0}$ is defined as follows for tabular definitions and does not include the electrolyte concentration terms:

$$\overline{I_0} = F k_c^{\alpha_a} k_a^{\alpha_c} C_{s,P}^{\max} \left(1 - \widehat{C}_{s,P}^{\text{surf}}\right)^{\alpha_a} \left(\widehat{C}_{s,P}^{\text{surf}}\right)^{\alpha_c}.$$

Alternatively, you can define your own custom Butler-Volmer intercalation current based on alternative formulations using user subroutine *UECHEMECDBV*.

Input File Usage: Use the following options to define the parameters used in the Butler-Volmer computation:

**MATERIAL*
**PARAMETER TABLE*,
TYPE=*"ABQ_EChemPET_Electrode_Particle_ButlerVolmer"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_ButlerVolmer"

k_c, k_a, α_c, α_a, C_e^{ref}, R_{SEI}, $Flag_{stressCoef}^{compute}$, a_s

Use the following options to specify a tabular form of $\overline{I_0}$ as a function of the normalized surface particle concentration and, optionally, temperature and field variables:

**MATERIAL*
**PROPERTY TABLE*,
TYPE=*"ABQ_EChemPET_Electrode_Particle_CurrXchgDens_Tabular"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_CurrXchgDens_Tabular"

$\overline{I_0}$, $\widehat{C}_{s,P}^{\text{surf}}$, θ, **FV**

Use the following options to define the coefficients of the Arrhenius model:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Arrhenius"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_BVArrhenius"
E_a, θ_{ref}

Use the following option to activate the definition of the Butler-Volmer intercalation current using user subroutine *UECHEMECDBV*:

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_IBVUser*"
1

Use the following option to optionally define a distribution (see *Distribution Definition*) of particle surface area per unit volume, a_s that changes from one element to another:

**MATERIAL*
**PARAMETER TABLE*,
TYPE="*ABQ_EChemPET_Electrode_Particle_a_sDistribution*",
LABEL="ABQ_EChemPET_<*Electrode name*>_Particle_a_sDistribution"
"Distribution Table Name"

Specifying the Joule Heat Fraction in the Different Phases

You specify the contribution of the different phases to the overall Joule heat fraction using two parameter tables, one for the macroscale and one for the microscale. The entropy table is used to define the derivative of the open circuit potential with respect to temperature. This term is one of the five terms that contribute to the overall Joule heat fraction.

Input File Usage: Use the following options to define the Joule heat fraction in the macroscale and microscale:

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemPET_Electrode_PowerLoss",
LABEL="ABQ_EChemPET_<*Electrode name*>_PowerLoss"

β_3, β_4, β_5

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemPET_Electrode_Particle_PowerLoss",

LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>_PowerLoss"

β_1, β_2

*MATERIAL
*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_dUdTEntropy_Tabular",
LABEL="ABQ_EChemPET_<*Electrode name*>_<*Particle name*>dUdTEntropy_Tabular"

$\frac{\partial U}{\partial \theta}$, $\widehat{C}_{s,P}$

Arrhenius Temperature Dependency

An Arrhenius form of temperature dependence is also supported for the definition of many of the material properties. The Arrhenius form of temperature dependence provides a scaling of the corresponding material property given as:

$$X(\theta) = X_0 e^{-\frac{E_a}{R}\left(\frac{1}{(\theta-\theta^z)}-\frac{1}{(\theta_{\text{ref}}-\theta^z)}\right)},$$

where

E_a
is the activation energy,

R
is the universal gas constant,

θ^z
is the absolute zero temperature,

θ_{ref}
is the reference temperature,

X_0

represents the value of material property at the reference temperature, and

X

is the resulting value after accounting for Arrhenius temperature dependency.

The Arrhenius form is ignored when you specify an explicit tabular temperature dependency in the property table definitions for the different properties (using the temperature parameter).

Input File Usage: Use the following options to define the coefficients of the Arrhenius model:

**MATERIAL*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Arrhenius"*
E_a, θ_{ref}

Universal Constants

You must define universal constants such as the Faraday number, gas constant, Avogadro's number, Boltzmann number, absolute zero temperature, and the elementary charge number.

Input File Usage: **PHYSICAL CONSTANTS*

Include File for Property and Parameter Table Definitions

The type definitions for the parameter and property tables described in the previous sections are described in *Parameter Table Type Reference* and *Property Table Type Reference* and available as an include file (see *Including Model or History Data from an External File*). You can use the **abaqus fetch** utility to obtain the file containing the type definitions of the parameter and property tables used by the electrochemistry framework.

```
abaqus fetch job=ABQ_EchemPET_Types.inp
```

Elements

The simultaneous solution in a coupled thermal-electrochemical analysis requires the use of elements that have electric potential in the solid (degree of freedom 9), temperature (degree of freedom 11), electric potential in the electrolyte (degree of freedom 32), and ion concentration in the electrolyte (degree of freedom 33) as nodal variables. The coupled thermal-electrochemical elements are available in Abaqus/Standard only in three dimensions (see *Coupled Thermal-Electrochemical Elements*).

Output

In addition to the output quantities available for the coupled thermal-electric procedure, you can request the following output variables in a coupled thermal-electrochemical analysis.

Nodal output variables:

EPOT	Electric potential in the solid phase.
EPOTE	Electric potential in the fluid (electrolyte).
NNCE	Ion concentration in the fluid (electrolyte).
RECURE	Reaction current in the fluid (electrolyte).

RFLCE Reaction ion concentration in the fluid (electrolyte).

Element integration point variables are available as described in *Coupled Thermal-Electrochemical Elements* and *Coupled Thermal-Electrochemical Elements, Particle Type–Related Variables*.

Whole model variables are available as described in *Total Amount Output Quantities*.

Input File Template

```
*HEADING
*INCLUDE, INPUT=ABQ_EchemPET_Types.inp
*MATERIAL, NAME=ABQ_EChemPET_matName
*DEPVAR
*CONDUCTIVITY
*DENSITY
*SPECIFIC HEAT
*POROUS ELECTRODE THEORY
*PARAMETER TABLE
*PROPERTY TABLE
…
*STEP
*COUPLED THERMAL-ELECTROCHEMICAL
*STEP CONTROL
*BOUNDARY
*CECURRENT
*FILM
*END STEP
```

References

- Ebner, M., , and V. Wood, “Tool for Tortuosity Estimation in Lithium-Ion Battery Porous Electrodes,” *Journal of the Electrochemical Society*, vol. 162, no. 2, 2015.
- Ecker, M., S. Kabitz, I. Laresgoiti, and D. Sauer, “Parameterization of a Physico-Chemical Model of a Lithium-Ion Battery II. Model Validation,” *Journal of the Electrochemical Society*, vol. 162, no. 9, 2015.
- Ecker, M., T. K. D. Tran, P. Dechent, S. Kabitz, A. Warnecke, and D. Sauer, “Parameterization of a Physico-Chemical Model of a Lithium-Ion Battery I. Determination of Parameters,” *Journal of the Electrochemical Society*, vol. 162, no. 9, 2015.
- Kumaresan, K., G. Sikha, and R E. White, “Thermal Model for a Li-Ion Cell,” *Journal of the Electrochemical Society*, vol. 155, no. 2, 2008.
- Liu, B., X. Wang, H. S. Chen, S. Chen, H. Yang, J. Xu, H. Jiang, B. Liu, and D. N. Fang, “A Simultaneous Multiscale and Multiphysics Model and Numerical Implementation of a Core-Shell Model for Lithium-Ion Full-Cell Batteries,” *Journal of Applied Mechanics*, vol. 86, 2019.
- Newman, J., , and K. E. Thomas-Alyea, “Electrochemical Systems,” *Wiley-Interscience, Third Edition*, 2004.
- Wu, B., “Modeling and Design of Lithium-Ion Batteries: Mechanics and Electrochemistry,” *Doctoral Dissertation, University of Michigan*, 2019.

Fully Coupled Thermal-Electrochemical-Structural Analysis

Products: Abaqus/Standard

Overview

A fully coupled thermal-electrochemical-structural analysis is a monolithic solve of the coupled thermal-displacement (see *Fully Coupled Thermal-Stress Analysis in Abaqus/Standard*) and coupled thermal-electrochemical (see *Coupled Thermal-Electrochemical Analysis*) analyses.

A fully coupled thermal-electrochemical-structural analysis:

- is used when the mechanical, thermal, electrical, and ion concentration fields affect each other strongly; and
- requires the use of coupled thermal-electrochemical-structural elements.

References:

- **COUPLED TEMPERATURE-DISPLACEMENT*
- *Parameter Table Type Reference*
- *Property Table Type Reference*
- *Fully coupled thermal-electrochemical-structural analysis in sodium ion batteries*
- *Fully coupled thermal-electrochemical-structural analysis in lithium-ion batteries*

Typical Application

The primary example of a battery electrochemistry application is the charging and discharging of lithium-ion battery cells. During the charging cycle, the lithium ions are extracted (deintercalated) from the active particles of the positive electrode (cathode). This process results in a reduction of the volume of the active particles. The ions move through the electrolyte by migration and diffusion from the positive electrode to the negative electrode (anode). At the anode, the ions intercalate into the active particles. This process results in an increase of the volume of the active particles and induces significant variation in tortuosities on both electrodes, thus strongly influencing the overall electrochemical behavior. Heat is generated during the flow of current in the solid and liquid phases, flow of current in the solid-liquid interface, and flow of ions in the electrolyte. During discharging, the cycle is reversed.

Coupled Thermal-Electrochemical-Structural Analysis

Rechargeable lithium-ion batteries are widely used in a variety of applications, including portable electronic devices and electric vehicles. The performance of a battery highly depends on the effects of repeated charging and discharging cycles, which can cause the degradation of the battery capacity over time. The porous electrode theory (*Newman et al., 2004*) is commonly accepted as the leading method for modeling the charge-discharge behavior of lithium-ion cells. The method is based on a homogenized Newman-type approach that does not consider the details of the pore geometry. The porous electrode theory is based on a concurrent solution of a highly coupled multiphysics-multiscale formulation. For further details on the thermal-electrochemical analysis, see *Coupled Thermal-Electrochemical Analysis*.

In some applications, a detailed understanding of the effects of the thermal-electrochemical fields on the mechanical state (deformations) of the lithium-ion cell is important and can have a strong influence on the overall performance of the cell. Battery performance characteristics (such as energy storage capacity and discharge voltages) can degrade with deformations caused by particle swelling and thermal strains. Large deformations

can cause failure of the separator, resulting in thermal runaways in batteries. In such applications, the coupling between temperature and displacement fields might be due to temperature-dependent material properties, internal heat generation, or thermal expansion. The volume changes in the active electrode particles resulting from the intercalation/deintercalation of lithium ions are modeled as particle concentration–dependent eigenstrains at the macroscale level. These volume changes affect the displacement field in the cell and porosity and tortuosity evolutions. In addition, they generate a convective transport term in the electrolyte that influences the movement of lithium ions through the electrolyte.

Governing Equations

The governing equations for the thermal-electrochemical process are based on the porous electrode theory and are described in detail in *Governing Equations*. The governing equations for particle swelling are also described in *Particle Swelling*. The volumetric strain that is computed based on particle swelling affects the macroscale quantities, such as the solid volume fraction, ε_s; porosities; tortuosities; and eigenstrain.

Eigenstrain (also referred to as inherent strain, assumed strain, or "stress-free" strain) is an engineering concept used to account for all sources of inelastic deformation that lead to residual stresses and distortions in manufactured components. For example, thermal strains are an example of eigenstrains. Abaqus solves for mechanical equilibrium of all internal and external forces in the system.

In linear elastic deformation, the stress induced by an eigenstrain can be represented as

$$\boldsymbol{\sigma} = \mathbf{D}^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^*) = \mathbf{D}^{el} : \boldsymbol{\varepsilon}^{el},$$

where

$\boldsymbol{\sigma}$	is the Cauchy stress;
$\mathbf{D}^{el}$	is the elasticity matrix;
$\boldsymbol{\varepsilon}$	is the total strain;
$\boldsymbol{\varepsilon}^*$	is the eigenstrain; and
$\boldsymbol{\varepsilon}^{el}$	is the elastic strain.

Using constitutive equations (such as that shown above), eigenstrains can be used to compute the stresses coming from mechanical, thermal, and microstructural sources.

It is possible to include the effects of particle swelling in both the coupled thermal-electrochemical and the coupled thermal-electrochemical-structural analyses. In a purely coupled thermal-electrochemical analysis, particle swelling results in convection of the electrolyte, which leads to changes in the porosity and the tortuosity of the electrodes. In a coupled thermal-electrochemical-structural analysis, particle swelling also results in mechanical deformations that are modeled as eigenstrains and results in stresses in the medium. Such mechanical deformations can also have an impact on the performance of a battery cell.

Fully Coupled Solution Scheme

A fully coupled solution scheme is needed when the stress analysis is dependent on the other fields involved in an electrochemical analysis, such as temperature, electric potentials in the solid and electrolyte, and ion concentration. In Abaqus/Standard, the temperature is integrated in time using a backward-difference scheme. The nonlinear coupled system is solved using Newton's method. The coupled thermal-electrochemical-structural analysis in Abaqus uses an exact implementation of Newton’s method, leading to an unsymmetric Jacobian matrix in the form:

$$\begin{bmatrix} K_{uu} & K_{u\varphi_s} & K_{u\theta} & K_{u\varphi_e} & K_{uC_e} \\ K_{\varphi_s u} & K_{\varphi_s\varphi_s} & K_{\varphi_s\theta} & K_{\varphi_s\varphi_e} & K_{\varphi_s C_e} \\ K_{\theta u} & K_{\theta\varphi_s} & K_{\theta\theta} & K_{\theta\varphi_e} & K_{\theta C_e} \\ K_{\varphi_e u} & K_{\varphi_e\varphi_s} & K_{\varphi_e\theta} & K_{\varphi_e\varphi_e} & K_{\varphi_e C_e} \\ K_{C_e u} & K_{C_e\varphi_s} & K_{C_e\theta} & K_{C_e\varphi_e} & K_{C_e C_e} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta\varphi_s \\ \Delta\theta \\ \Delta\varphi_e \\ \Delta C_e \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_{\varphi_s} \\ R_\theta \\ R_{\varphi_e} \\ R_{C_e} \end{Bmatrix}.$$

Steady-State Analysis

Steady-state analysis provides the steady-state solution by neglecting the transient terms in the continuum scale equations. It can be used to achieve a balanced initial state or to assess conditions in the cell after a long storage period.

In the thermal equation, the internal energy term in the governing heat transfer equation is omitted. Similarly, the transient term is omitted in the diffusion equations for the lithium ion concentration in the electrolyte. Electrical transient effects are not included in the equations because they are very rapid compared to the characteristic times of thermal and diffusion effects. A steady-state analysis has no effect on the microscale solution; the transient terms are always considered in the solution of the lithium concentration in the solid particle.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, STEADY STATE

Transient Analysis

In a transient analysis, the transient effects in the heat transfer and diffusion equations are included in the solution. Electrical transient effects are always omitted because they are very rapid compared to the characteristic times of thermal and mass diffusion effects.

Input File Usage: **COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL

Spurious Oscillations due to Small Time Increments

The integration procedure Abaqus/Standard uses for the transient heat transfer equation introduces a relationship between the minimum usable time increment and the element size. Time increments below this minimum can lead to spurious oscillations in the temperature field. By default, Abaqus/Standard uses nodal integration for the heat capacity term for first-order elements in a transient analysis, resulting in a lumped treatment for this term. This treatment eliminates nonphysical oscillations; however, it can lead to locally inaccurate solutions, especially in terms of the heat flux for small time increments. If smaller time increments are required, you should use a finer mesh in regions where the temperature changes occur. For additional details for time integration involving temperature degree of freedom, see *Spurious Oscillations due to Small Time Increments* in *Uncoupled Heat Transfer Analysis*.

Controlling the Analysis Based on Solution Variables

A battery analysis typically consists of four (or more) steps, in the following order:

- A charging step, in which the battery is subjected to a constant charging current. The voltage is monitored,and the step is terminated when the voltage increases to a specified cut-off value.
- A constant voltage step, in which the battery is subjected to a constant prescribed voltage. The reaction current is monitored, and the step is terminated when the reaction current drops below a specified threshold value.
- A resting step, in which the battery is not subjected to any prescribed current or voltage.

- A discharge step, in which a constant current is drawn from the battery. The voltage is monitored, and the step is terminated when the voltage drops to a specified cut-off value.

You can use the step control mechanism (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) together with sensors (see *Defining Sensors*) to model the workflow described above. In particular, you can associate the history output of the voltage or the current at specific locations in the model with sensor definitions, and terminate a step when the sensor reaches a predefined cut-off value or threshold.

Initial Conditions

By default, the initial values of electric potential in the solid, temperature, electric potential in the electrolyte, and ion concentration of all nodes are set to zero. You can specify nonzero initial values for the primary solution variables (see *Initial Conditions*).

Boundary Conditions

You can prescribe the following boundary conditions:

- Displacement degrees of freedom (degrees of freedom 1, 2, and 3).
- Electric potential in the solid, $\varphi_s = \varphi_s(\mathbf{x}, t)$ (degree of freedom 9).
- Electric potential in the electrolyte, $\varphi_e = \varphi_e(\mathbf{x}, t)$ (degree of freedom 32).
- Temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11).
- Ion concentration in the electrolyte, $C_e = C_e(\mathbf{x}, t)$ (degree of freedom 33) at the nodes.

You can specify boundary conditions as functions of time by referring to amplitude curves.

A boundary without any prescribed boundary conditions corresponds to an insulated (zero flux) surface.

The typical boundary condition consists of only grounding (setting to zero) the solid electric potential at the anode. Thermal boundary conditions vary.

Loads

You can apply mechanical, thermal, electrical, and electrochemical loads in a coupled thermal-electrochemical-structural analysis.

You can prescribe the following types of mechanical loads (as described in *Concentrated Loads* and *Distributed Loads*):

- Concentrated nodal forces on displacement degrees of freedom.
- Distributed forces.

You can prescribe the following types of thermal loads (as described in *Thermal Loads*):

- Concentrated heat flux.
- Body flux and distributed surface flux.
- Convective film and radiation conditions.

You can prescribe the following types of electrical loads on the solid (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of electrical loads on the electrolyte (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of ion concentration loads (as described in *Thermal Loads*):

- Concentrated flux.
- Distributed body flux.

The typical loads include specification of a solid electric flux (current) at the cathode. Thermal boundary conditions vary but typically include convective film on the exterior surfaces. Customarily, no loads are applied on the concentrations and electrolyte potential.

Predefined Fields

Predefined temperature fields are not allowed in coupled thermal-electrochemical-structural analyses. Instead, you can use boundary conditions to prescribe degree of freedom 11. You can specify other predefined field variables in a fully coupled thermal-electrochemical-structural analysis. These values affect only field variable–dependent material properties, if any.

Material Options

The material definition in a fully coupled thermal-electrochemical-structural analysis must include thermal, electrical, electrochemical, and mechanical properties.

The electrochemistry framework requires that the material definition contain the complete specification of properties required for the porous electrode theory, as described in *Material Options*. In addition, the material name must begin with "ABQ_EChemPET_" to enable the coupled micro-macro solution at the different electrodes. Special-purpose parameter and property tables of type names starting with “ABQ_EChemPET_” are required in these material definitions (see *Parameter Table Type Reference* and *Property Table Type Reference*). For more details about the material definitions for the thermal-electrochemical behavior, see *Material Options*.

You can define anisotropic swelling of the particle to be used in eigenstrain computations.

Input File Usage: Use the following options to specify properties for the porous electrode theory:

**MATERIAL*, NAME=ABQ_EChemPET_*name*
**POROUS ELECTRODE THEORY*
**ELASTIC*

Use the following options to define anisotropic swelling of the particle:

**MATERIAL*, NAME=ABQ_EChemPET_*name*
**PARAMETER TABLE*, TYPE=*"ABQ_EChemPET_Electrode_Swelling"*,
LABEL="ABQ_EChemPET_<*Electrode name*>_Swelling"
X_{scale}, Y_{scale}, Z_{scale}

Elements

A fully coupled thermal-electrochemical-structural analysis requires the use of elements that have displacement (degrees of freedom 1, 2, 3), electric potential in the solid (degree of freedom 9), temperature (degree of freedom 11), electric potential in the electrolyte (degree of freedom 32), and ion concentration in the electrolyte (degree of freedom 33) as nodal variables. The coupled thermal-electrochemical-structural elements are available in Abaqus/Standard only in three dimensions (see *Coupled Thermal-Electrochemical-Structural Elements*).

A fully coupled thermal-electrochemical-structural analysis can also include regions modeled using the following categories of nonelectrochemistry elements: three-dimensional stress/displacement solid elements, three-dimensional stress/displacement shell elements, three-dimensional coupled temperature-displacement solid elements, three-dimensional coupled temperature-displacement shell elements, as well as rigid elements R3D3 and R3D4.

Output

In addition to the output quantities available for the coupled thermal-electric and the coupled thermal-electrochemical procedures, you can request the Abaqus/Standard output variables for mechanical degrees of freedom.

Input File Template

```
*HEADING
*INCLUDE, INPUT=ABQ_EchemPET_Types.inp
*MATERIAL, NAME=ABQ_EChemPET_matName
*DEPVAR
*ELASTIC
*CONDUCTIVITY
*DENSITY
*SPECIFIC HEAT
*POROUS ELECTRODE THEORY
*PARAMETER TABLE
*PROPERTY TABLE
…
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTROCHEMICAL
*STEP CONTROL
*BOUNDARY
*CECURRENT
*FILM
*END STEP
```

References

- Newman, J., and K. E. Thomas-Alyea, “Electrochemical Systems,” *Wiley-Interscience, Third Edition*, 2004.

Fully Coupled Thermal-Electrochemical-Structural–Pore Pressure Analysis

Products: Abaqus/Standard

Overview

You must use coupled structural-thermal-electrochemical–pore pressure elements in this type of analysis.

A fully coupled thermal-electrochemical-structural–pore pressure analysis is used for the analysis of battery electrochemistry applications that require solving simultaneously for the following:

- displacements,
- temperature,
- electric potentials in the solid electrodes,
- electric potential in the electrolyte,
- concentration of ions in the electrolyte,
- concentration in the solid particles used in the electrodes, and
- fluid pore pressure mediated electrolyte flow with or without convective effects. Particle swelling and element deformation can affect the electrolyte convective flow and fluid pressure.

A complete understanding of battery electrochemistry requires modeling the mutual influence of the deformation of the porous electrodes and the flow of the ion-carrying electrolyte within the interconnected pores. This coupling is an important aspect of the battery multiphysics, and you can use the fully coupled thermal-electrochemical-structural–pore pressure procedure to better understand the influence of this coupling on the battery performance. A fully coupled thermal-electrochemical-structural–pore pressure analysis is a monolithic solve of the coupled pore fluid diffusion-displacement (see *Coupled Pore Fluid Diffusion and Stress Analysis*) and coupled thermal-electrochemical-structural (see *Fully Coupled Thermal-Electrochemical-Structural Analysis*) analyses.

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*
- **COUPLED TEMPERATURE-DISPLACEMENT*

Typical Application

The primary example of a fully coupled thermal-electrochemical-structural–pore pressure analysis in a battery electrochemistry application is the charging and discharging of lithium ion battery cells. During the charging cycle, the lithium ions are extracted (deintercalated) from the active particles of the positive electrode (cathode). This process results in a reduction of the volume of the active particles, which in turn leads to an increase in porosity as well as flow of electrolyte into the cathode. The ions move through the electrolyte by migration and diffusion from the positive electrode to the negative electrode (anode). At the anode, the ions intercalate into the active particles. This process results in an increase of the volume of the active particles, which in turn leads to a decrease in porosity as well as flow of electrolyte out of the anode. Heat is generated during the flow of current in the solid and liquid phases, flow of current in the solid-liquid interface, and flow of ions in the electrolyte. During discharging, the cycle is reversed.

Coupled Thermal-Electrochemical-Structural–Pore Pressure Analysis

Rechargeable lithium-ion batteries are widely used in a variety of applications, including portable electronic devices and electric vehicles. The performance of a battery highly depends on the effects of repeated charging and discharging cycles, which can cause the degradation of the battery capacity over time. The porous electrode theory (*Newman et al., 2004*) is commonly accepted as the leading method for modeling the charge-discharge behavior of lithium-ion cells. The method is based on a homogenized Newman-type approach that does not consider the details of the pore geometry. The porous electrode theory is based on a concurrent solution of a highly coupled multiphysics-multiscale formulation. For more information, see *Coupled Thermal-Electrochemical Analysis*.

In some applications, a detailed understanding of the effects of the thermal-electrochemical fields on the mechanical state (deformations) of the lithium-ion cell, electrolyte fluid pressures, and electrolyte movement can be important to understand their influence on the overall performance of the cell. Battery performance characteristics (such as energy storage capacity and discharge voltages) can degrade with deformations caused by particle swelling, thermal strains, and ion depletion. Large deformations can cause failure of the separator, resulting in thermal runaways in batteries. In such applications, the coupling between temperature and displacement fields might be due to temperature-dependent material properties, internal heat generation, or thermal expansion. The volume changes in the active electrode particles resulting from the intercalation/deintercalation of lithium ions are modeled as particle concentration–dependent eigenstrains at the macroscale level. These volume changes affect the displacement field in the cell and porosity and tortuosity evolutions. In addition, they generate a convective transport term in the electrolyte that influences the movement of lithium ions through the electrolyte.

The effects of fluid flow and fluid pressure are important to determine localized regions of low electrolyte concentrations within the battery that can degrade battery performance. It is possible to model the different regions in the battery as either fully saturated or partially saturated to get a better understanding of electrolyte flow effects during a charge-discharge cycle.

Governing Equations

The governing equations for the thermal-electrochemical process are based on the porous electrode theory and are described in detail in *Governing Equations*. The governing equations for particle swelling are also described in *Particle Swelling*. The volumetric strain at the microscale, caused by particle swelling, results in macroscopic eigenstrains (also referred to as inherent strain, assumed strain, or "stress-free" strain) that affects the deformation of the porous electrodes, solid volume fraction, ε_s, pore fluid pressure in the electrolyte, and electrolyte flow. Abaqus treats the continuum (porous electrodes containing electrolyte) as a multiphase material and adopts an effective stress principle to describe its behavior (see *Effective stress principle for porous media*). In this approach, the total stress at a material point is assumed to be made up of an effective stress, $\overline{\sigma}$, which represents the stress in the skeleton of the porous electrode, and an average pressure stress, u_w, in the electrolyte, such that:

$$\overline{\sigma} = \sigma + s u_w I.$$

The above expression assumes partially saturated electrolyte flow, with saturation level, s, where $s = 1.0$ for fully saturated flow and $s < 1.0$ represents partially saturated flow. The response of such a porous medium is based on satisfying:

- mechanical equilibrium based on all internal and external forces in the system, as well as
- fluid continuity for the electrolyte phase.

The porous medium is modeled with a Lagrangian finite element mesh that keeps track of the evolving volume fractions of the solid and electrolyte phases at each material point, as well as the flow of the electrolyte phase. The electrolyte is assumed to be relatively incompressible. The effective response of the porous medium can be described in terms of any mechanical constitutive model in Abaqus. For the special case of a linear elastic effective response, the effective stress induced by an eigenstrain can be represented as:

$$\overline{\boldsymbol{\sigma}} = \mathbf{D}^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{*}) = \mathbf{D}^{el} : \boldsymbol{\varepsilon}^{el},$$

where

$\overline{\boldsymbol{\sigma}}$	is the effective Cauchy stress;
$\mathbf{D}^{el}$	is the elasticity matrix;
$\boldsymbol{\varepsilon}$	is the total strain;
$\boldsymbol{\varepsilon}^{*}$	is the eigenstrain; and
$\boldsymbol{\varepsilon}^{el}$	is the elastic strain.

The continuity equation for the electrolyte flow is described in *Continuity statement for the wetting liquid phase in a porous medium*. The formulation of the continuity equation is in terms of pore pressure as the basic nodal variable (degree of freedom 8 at each node). The conjugate flux is the nodal volumetric flow rate.

The ion flux equation (see *Diffusion and Migration of Lithium Ions in the Liquid Phase*) can optionally include advection contributions resulting from the velocity field of the electrolyte. If advection effects are included in the model, any electrolyte flux at the external boundaries is associated with a corresponding ion flux (that is, the electrolyte leaves or enters the domain with ions). If advection effects are not included in the model, only "clean electrolyte" (that is, electrolyte without any ions) can leave or enter through the external boundaries.

Fully Coupled Solution Scheme

A fully coupled solution scheme is required when the stress analysis is strongly dependent on the other fields involved in an electrochemical analysis, such as temperature, pore pressure, electric potentials in the solid and electrolyte, and ion concentration. In Abaqus/Standard, the temperature is integrated in time using a backward-difference scheme. The nonlinear coupled system is solved using Newton's method. The coupled thermal-electrochemical-structural–pore pressure analysis in Abaqus uses an exact implementation of Newton's method, leading to an unsymmetric Jacobian matrix in the form:

$$\begin{bmatrix} K_{uu} & K_{uu_w} & K_{u\varphi_s} & K_{u\theta} & K_{u\varphi_e} & K_{uC_e} \\ K_{u_wu} & K_{u_wu_w} & K_{u_w\varphi_s} & K_{u_w\theta} & K_{u_w\varphi_e} & K_{u_wC_e} \\ K_{\varphi_su} & K_{\varphi_su_w} & K_{\varphi_s\varphi_s} & K_{\varphi_s\theta} & K_{\varphi_s\varphi_e} & K_{\varphi_sC_e} \\ K_{\theta u} & K_{\theta u_w} & K_{\theta\varphi_s} & K_{\theta\theta} & K_{\theta\varphi_e} & K_{\theta C_e} \\ K_{\varphi_eu} & K_{\varphi_eu_w} & K_{\varphi_e\varphi_s} & K_{\varphi_e\theta} & K_{\varphi_e\varphi_e} & K_{\varphi_eC_e} \\ K_{C_eu} & K_{C_eu_w} & K_{C_e\varphi_s} & K_{C_e\theta} & K_{C_e\varphi_e} & K_{C_eC_e} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta \mathrm{u}_w \\ \Delta\varphi_s \\ \Delta\theta \\ \Delta\varphi_e \\ \Delta \mathrm{C}_e \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_{u_w} \\ R_{\varphi_s} \\ R_\theta \\ R_{\varphi_e} \\ R_{C_e} \end{Bmatrix}.$$

Steady-State Analysis

Steady-state analysis provides the steady-state solution by neglecting the transient terms in the continuum scale equations. It can be used to achieve a balanced initial state or to assess conditions in the cell after a long storage period.

A steady-state analysis neglects the transient terms in all the macroscopic field equations. These include:

- internal energy term in heat transfer;
- transient term in the diffusion equation for the lithium ion concentration in the electrolyte;
- electrical transients in the conduction equations for both the solid and electrolyte phases (characteristic time scales associated with the electrical transient effects are much smaller compared to the characteristic times of thermal and ion-diffusion); and
- electrolyte flow continuity equation. When the modeling of electrolyte flow includes gravity effects, a steady-state step ensures the loads and initial stresses equilibrate exactly.

A steady-state analysis has no effect on the microscale solution; the transient terms are always included in the solution of the lithium concentration in the solid particle.

Input File Usage: Use the following option to define a steady-state fully coupled thermal-electrochemical-structural–pore pressure analysis that considers the fluid velocity effects:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, PORE PRESSURE, STEADY STATE, ADVECTION=YES

Use the following option to define a steady-state fully coupled thermal-electrochemical-structural–pore pressure analysis that ignores the fluid velocity effects:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, PORE PRESSURE, STEADY STATE, ADVECTION=NO

Transient Analysis

In a transient analysis, the transient effects in the heat transfer, pore fluid continuity, and ion-diffusion equations are included in the solution. Electrical transient effects are always omitted because they are rapid compared to the characteristic times of thermal and ion-diffusion effects.

Input File Usage: Use the following option to define a transient fully coupled thermal-electrochemical-structural–pore pressure analysis that considers the fluid velocity effects:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, PORE PRESSURE, ADVECTION=YES

Use the following option to define a transient fully coupled thermal-electrochemical-structural–pore pressure analysis that ignores the fluid velocity effects:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, PORE PRESSURE, ADVECTION=NO

Ambient Ion Concentration

In a fully coupled thermal-electrochemical-structural–pore pressure analysis, you can model the presence of an external reservoir of fluid electrolyte with a prescribed ion concentration value of C_e^{amb}.

When you model fluid velocity effects in conjunction with the presence of an external reservoir:

- If the pore pressure inside the domain is higher than the prescribed pore pressure at a boundary node, fluid electrolyte leaves the active domain through the node. The exiting fluid carries an ionic concentration of value equal to that of the ion concentration degree of freedom at that boundary node at that instance in time.
- If the pore pressure inside the domain is lower than the prescribed pore pressure at the boundary node, fluid electrolyte enters the active domain through the node. The entering fluid carries an ionic concentration equal to the prescribed ambient ion concentration value of C_e^{amb}.

Input File Usage: Use the following option to define a transient fully coupled thermal-electrochemical-structural–pore pressure analysis that considers the fluid velocity effects in conjunction with the presence of an external reservoir:

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, PORE PRESSURE, ADVECTION=YES, AMBIENT ION CONCENTRATION=C_e^{amb}

Spurious Oscillations Due to Small Time Increments

The integration procedure Abaqus/Standard uses for fluid flow analysis with the pore pressure degree of freedom introduces a relationship between the minimum usable time increment and the element size. If time increments smaller than these values are used, spurious oscillations can appear in the solution.

The integration procedure Abaqus/Standard uses for the transient heat transfer equation also results in similar limitations on the minimum usable time increment. By default, Abaqus/Standard uses nodal integration for the heat capacity term for first-order elements in a transient analysis, resulting in a lumped treatment for this term. This treatment eliminates nonphysical oscillations; however, it can lead to locally inaccurate solutions, especially in terms of the heat flux for small time increments. If smaller time increments are required, you should use a finer mesh in regions where the temperature changes occur.

For more information about time integration involving the pore pressure and temperature degrees of freedom, see *Spurious Oscillations due to Small Time Increments* in *Uncoupled Heat Transfer Analysis*.

Controlling the Analysis Based on Solution Variables

A battery analysis typically consists of four (or more) steps in the following order:

- A charging step, in which the battery is subjected to a constant charging current. The voltage is monitored, and the step is terminated when the voltage increases to a specified cut-off value.
- A constant voltage step, in which the battery is subjected to a constant prescribed voltage. The reaction current is monitored, and the step is terminated when the reaction current drops below a specified threshold value.
- A resting step, in which the battery is not subjected to any prescribed current or voltage .
- A discharge step, in which a constant current is drawn from the battery. The voltage is monitored, and the step is terminated when the voltage drops to a specified cut-off voltage.

You can use the step control mechanism (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) together with sensors (see *Defining Sensors*) to model the workflow described above. In particular, you can associate the history output of the voltage or the current at specific locations in the model with sensor definitions, and terminate a step when the sensor reaches a predefined cut-off value or threshold.

Initial Conditions

By default, the initial values of pore pressure, electric potential in the solid and electrolyte, temperature, and ion concentration of all nodes are set to zero. You can specify nonzero initial values for the primary solution variables (see *Initial Conditions*).

Boundary Conditions

You can prescribe the following boundary conditions:

- Displacement (degrees of freedom 1, 2, and 3)
- Pore pressure, $u_w = u_w(\mathbf{x}, t)$ (degree of freedom 8)

- Electric potential in the solid, $\varphi_s = \varphi_s(\mathbf{x}, t)$ (degree of freedom 9)
- Electric potential in the electrolyte, $\varphi_e = \varphi_e(\mathbf{x}, t)$ (degree of freedom 32)
- Temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11)
- Ion concentration in the electrolyte, $C_e = C_e(\mathbf{x}, t)$ (degree of freedom 33) at the nodes

You can specify boundary conditions as functions of time by referring to amplitude curves (see *Amplitude Curves*).

A boundary without any prescribed boundary conditions corresponds to an insulated (zero flux) surface.

The typical boundary condition consists of only grounding (setting to zero) the solid electric potential at the anode. Thermal boundary conditions vary.

Loads

You can apply mechanical, pore fluid pressure, thermal, electrical, and electrochemical loads in a coupled thermal-electrochemical-structural–pore pressure analysis.

You can prescribe the following types of mechanical loads (as described in *Concentrated Loads* and *Distributed Loads*):

- Concentrated nodal forces on displacement degrees of freedom.
- Distributed forces.

You can prescribe the following types of pore pressure loads (as described in *Pore Fluid Flow*):

- Distributed pressure forces or body forces. The distributed load types available with particular elements are described in the *Abaqus Elements Guide*. The magnitude and direction of gravitational loading are usually defined by using the gravity distributed load type.
- Pore fluid flow.

You can prescribe the following types of thermal loads (as described in *Thermal Loads*):

- Concentrated heat flux.
- Body flux and distributed surface flux.
- Convective film and radiation conditions.

You can prescribe the following types of electrical loads on the solid (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of electrical loads on the electrolyte (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of ion concentration loads (as described in *Thermal Loads*):

- Concentrated flux.
- Distributed body flux.

The typical loads include specification of an electric flux (current) in the solid phase of the cathode. Thermal boundary conditions vary but typically include convective film on the exterior surfaces. Typically, no loads are applied on the concentrations and the electrolyte potential fields.

Predefined Fields

Predefined temperature fields are not allowed in coupled thermal-electrochemical-structural–pore pressure analyses. Instead, you can use boundary conditions to prescribe degree of freedom 11 for temperature. You can specify other predefined field variables in a fully coupled thermal-electrochemical-structural–pore pressure analysis. These values affect only field variable–dependent material properties, if any.

Material Options

In a coupled thermal-electrochemical-structural-pore pressure analysis, you can specify the effective response of the medium in terms of any available mechanical constitutive behavior. The thermal and electrical properties for both the solid and the electrolyte phases are also active during this analysis.

The electrochemistry framework requires that the material definition contain the complete specification of properties required for the porous electrode theory, as described in *Coupled Thermal-Electrochemical Analysis*. In addition, the material name must begin with "ABQ_EChemPET_" to enable the coupled micro-macro solution at the different electrodes. Special-purpose parameter and property tables of type names starting with "ABQ_EChemPET_" are required in these material definitions (see *Parameter Table Type Reference* and *Property Table Type Reference*). For more information about the material definitions for the thermal-electrochemical behavior, see *Material Options*.

In problems formulated in terms of total pore pressure, you must include the density of the dry material in the material definition (see *Density*). You must specify the permeability of the porous medium including, optionally, its dependence on the void ratio, saturation, and the flow velocity (see *Permeability*). You must also define the specific weight of the electrolyte as part of the material permeability definition. You can define the compressibility of the solid grains and of the electrolyte for both fully and partially saturated flow problems (see *Elastic Behavior of Porous Materials*). If you do not specify the compressibility of the solid and the liquid phases, they are assumed to be fully incompressible.

For partially saturated flow, you must define the porous medium's absorption/exsorption behavior (see *Sorption*).

Elements

A fully coupled thermal-electrochemical-structural–pore pressure analysis requires the use of elements that have displacement (degrees of freedom 1, 2, and 3), pore pressure (degree of freedom 8), electric potential in the solid (degree of freedom 9), temperature (degree of freedom 11), electric potential in the electrolyte (degree of freedom 32), and ion concentration in the electrolyte (degree of freedom 33) as nodal variables. The coupled thermal-electrochemical-structural–pore pressure elements are available in Abaqus/Standard only in three dimensions (see *Coupled Thermal-Electrochemical-Structural–Pore Pressure Elements*).

A fully coupled thermal-electrochemical-structural–pore pressure analysis can also include regions modeled using the following categories of nonelectrochemistry elements: three-dimensional stress/displacement solid elements, three-dimensional stress/displacement shell elements, three-dimensional coupled temperature-displacement solid elements, three-dimensional coupled temperature-displacement shell elements, as well as rigid elements R3D3 and R3D4.

Output

In addition to the output variables available for the coupled thermal-electric, the coupled thermal-electrochemical, and coupled thermal-electrochemical structural procedures, you can request the Abaqus/Standard output variables related to pore pressure degrees of freedom (see *Output*).

Input File Template

```
*HEADING
*INCLUDE, INPUT=ABQ_EchemPET_Types.inp
*MATERIAL, NAME=ABQ_EChemPET_matName
*DEPVAR
*ELASTIC
*CONDUCTIVITY
*DENSITY
*SPECIFIC HEAT
*PERMEABILITY
*POROUS ELECTRODE THEORY
*PARAMETER TABLE
*PROPERTY TABLE
…
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTROCHEMICAL, PORE PRESSURE,
ADVECTION=YES

*BOUNDARY
*CECURRENT
*FILM
*END STEP
```

References

- Newman, J., , and K. E. Thomas-Alyea, “Electrochemical Systems,” *Wiley-Interscience, Third Edition*, 2004.

Modeling Solid Electrodes in Lithium Metal Batteries

Products: Abaqus/Standard

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*
- *Modeling a half-cell lithium metal anode battery with multiphysics Butler-Volmer surface loading*

Overview

Lithium metal batteries have a higher capacity and reduced size compared to porous electrode batteries.

Lithium Metal Batteries

A typical lithium metal battery has an anode that is made of solid lithium while the cathode is a porous electrode. The anode and cathode regions are separated by an electrically insulating porous separator, which allows ions to move across from anode to cathode and vice versa. *Figure 1* shows a typical configuration of a solid lithium battery. Lithium metal batteries tend to have a higher capacity and reduced size compared to batteries with all porous electrodes.

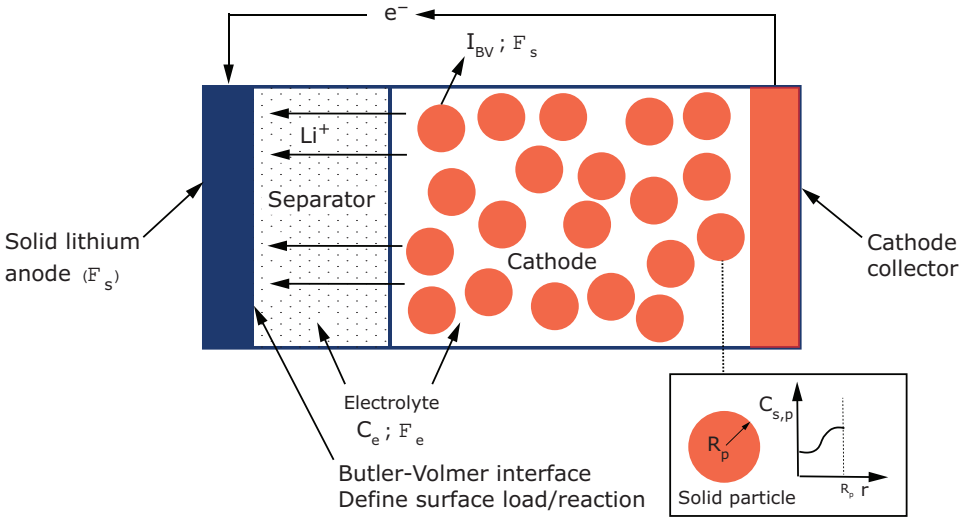


Figure 1: Schematic diagram of a solid lithium metal battery.

Since the anode is not porous, the Newman porous electrode theory framework is not required to model the anode. Instead, the behavior of the anode can be characterized using classical thermal-electrical-structural physics. The separator and cathode regions are modeled as porous electrodes as described in *Coupled Thermal-Electrochemical Analysis*. The Abaqus lithium metal battery models assume that the interface between the separator and the anode can be characterized using a surface-based Butler-Volmer load (see *Figure 1*) that accounts for fluid electric potential and ion concentration on the separator side and the solid electric potential on the anode side.

Governing Equations

The governing equations for the thermal-electrochemical process are based on the porous electrode theory and are described in detail in *Governing Equations* in *Coupled Thermal-Electrochemical Analysis*. At the interface between the separator and the solid lithium anode, a charge-transfer reaction occurs following a Butler-Volmer kinetics that is expressed using the following equations:

$$\mathrm{Li} - \Theta \leftrightarrow \mathrm{Li}^{+} + e^{-} + \Theta.$$

Here, $\mathrm{Li} - \Theta$ represents lithium in the solid electrode that, through the deintercalation reaction, transforms into a lithium ion in the electrolyte, Li^{+}, plus an electron, e^{-}, and leaves behind a vacancy (or intercalation site) in the solid, Θ.

The general form of the Butler-Volmer current expression is:

$$I_{\mathrm{BV}} = \overline{I_0}\left\{\exp\left[\frac{\alpha_a(zF\eta)}{R(\theta-\theta^z)}\right] - \exp\left[\frac{-\alpha_c(zF\eta)}{R(\theta-\theta^z)}\right]\right\}\left(\frac{C_e}{C_e^{\mathrm{ref}}}\right)^{\alpha_a},$$

where $\overline{I_0}$ and η are the exchange current density and overpotential, respectively. The overpotential, η, is defined as:

$$\eta = \Phi_s - \Phi_e - E_{\mathrm{ref}}(\theta).$$

In the above equations,

F
: is Faraday's constant;

R
: is the gas constant;

z
: is the charge number of the lithium ion battery;

θ
: is temperature;

θ^z
: is the absolute zero temperature;

α_c, α_a
: are the cathodic and anodic transfer coefficients, respectively;

C_e^{ref}
: is the reference value of lithium ion concentration in the electrolyte;

C_e
: is the ion concentration in the electrolyte from the separator side;

E_{ref}
: is the open circuit potential (OCP) as a function of θ. The typical value in a solid anode is zero.

ϕ_s
is the electric potential from the anode side; and

ϕ_e
is the fluid electric potential from the separator side.

The electrochemical interaction at the solid electrode-separator interface is represented on the solid electrode side as a surface current density of magnitude $-S_{fac}I_{BV}$. S_{fac} represents a dimensionless "surface factor" that captures the effects of surface irregularities that can result in the availability of a larger surface area over which the electrochemical interaction happens. On the separator side of the interface, the electrochemical interaction results in two separate surface fluxes, both applied on the electrolyte phase: a fluid current density of magnitude $S_{fac}I_{BV}$ and an ion flux of magnitude $\frac{S_{fac}I_{BV}}{F}$, where F is the Faraday constant. For more details on the electrochemical governing equation, see *Conduction of Electrons in the Solid Phase* and *Diffusion and Migration of Lithium Ions in the Liquid Phase*. However, an important distinction between the referenced governing equation sections and the current surface interaction load is that the conduction and the diffusion/migration equations in the referenced sections treat the effects of the Butler-Volmer kinetics as a volumetric source. Here, they contribute as surface fluxes for the solid lithium electrode battery.

Joule Heating at the Solid-Liquid Interface

The total heat generation at the interface of the solid electrode and the fluid electrolyte is attributed to two sources: flow of current at the solid-liquid interface and entropy generation. The amount of energy converted into heat from each of the source terms can be scaled using the conversion factors, β_1, β_2:

$$Q_{\text{tot}} = \beta_1 Q_1 + \beta_2 Q_2,$$

where

$Q_1 = S_{fac}I_{\text{BV}}\eta$
is the ohmic loss at the solid-liquid interface; and

$Q_2 = S_{fac}I_{\text{BV}}\left(\theta - \theta^z\right)\frac{\partial U}{\partial \theta}$
is the entropy generation, where $\frac{\partial U}{\partial \theta}$ is the derivative of the open circuit potential with respect to temperature that you can specify in tabular form.

Defining Properties for Butler-Volmer Kinetics

You must specify the constants required to define the Butler-Volmer kinetics for a given surface interaction using a table collection. The name of the table collection must be included in the definition of the corresponding Butler-Volmer surface interaction (*Defining a Butler-Volmer Surface Interaction*) .

You must define $\overline{I_0}$ and the open circuit potential for the anode in tabular form as a function of temperature.

Optionally, you can specify the temperature dependency of $\overline{I_0}$ using an Arrhenius definition (see *Arrhenius Temperature Dependency*). If Joule heating effects are important, you must also specify the Joule heating losses and the entropy table that is used to define the derivative of the open circuit potential with respect to temperature.

Input File Usage: Use a table collection definition to include all properties (listed below) required to define the Butler-Volmer kinetics:

*TABLE COLLECTION

Use the following options to define the coefficients of the Butler-Volmer kinetics equation:

*PARAMETER TABLE, TYPE="ABQ_EChem_LMB_ButlerVolmerInterf"

, , α_c, α_a, C_e^{ref}, , , S_{fac}, z

Use the following options to specify a tabular form of $\overline{I_0}$ as a function of temperature:

*PROPERTY TABLE, TYPE="ABQ_EChem_LMB_CurrXchgDens_Tabular"

$\overline{I_0}$, θ

Use the following options to define the coefficients of the Arrhenius model:

*PARAMETER TABLE, TYPE="ABQ_EChem_LMB_Arrhenius"

E_a, θ_{ref}

Use the following options to define the open circuit potential for a layer in tabular format:

*PROPERTY TABLE, TYPE="ABQ_EChem_LMB_OCPTabular"

E_{ref}, θ

Use the following options to define the Joule heat fraction at the interface:

*PARAMETER TABLE, TYPE="ABQ_EChem_LMB_Interface_PowerLoss"

β_1, β_2

*PROPERTY TABLE, TYPE="ABQ_EChem_LMB_dUdTEntropy_Tabular"

$\frac{\partial U}{\partial \theta}$, θ

Defining a Butler-Volmer Surface Interaction

Abaqus/Standard offers two options to define the Butler-Volmer surface interaction.

When the two regions (namely, the solid lithium anode and the porous separator) are modeled using matching meshes with shared nodes, you can define the Butler-Volmer interaction as a surface-based load on the facets of the anode elements.

If the meshes on the two sides of the interface between the solid anode and the separator are dissimilar and do not share nodes, you can use a surface interaction property to define an interface reaction behavior.

For both definitions you must specify the name of a table collection that includes the corresponding Butler-Volmer kinetics properties for the interface (*Defining Properties for Butler-Volmer Kinetics*).

Input File Usage: Use the following option to define a surface-based Butler-Volmer load:

*STEP,
*MULTIPHYSICS LOAD, DEFINITION=BUTLER-VOLMER

Use the following options to define an interface reaction behavior:

*SURFACE INTERACTION
*INTERFACE REACTION, TYPE=BUTLER-VOLMER
*CONTACT
*CONTACT PROPERTY ASSIGNMENT

Output

In addition to the output variables available for the coupled thermal-electric procedures, the coupled thermal-electrochemical procedures, the coupled thermal-electrochemical-structural procedures, and the thermal-electrochemical-structural-pore pressure procedures, you can request the Abaqus/Standard output variables related to surface heat generation at the solid-liquid interface. The surface heat fluxes are described in *Coupled Thermal-Electrochemical-Mechanical Analysis–Surface Facet Quantities*, and the total surface heat generation powers are described in *Coupled Thermal-Electrochemical-Mechanical Analysis*.

Modeling Aging in Batteries

Products: Abaqus/Standard

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*
- *Modeling a half-cell lithium metal anode battery with multiphysics Butler-Volmer surface loading*

Overview

Abaqus/Standard provides a set of capabilities to model battery aging.

Aging of lithium ion batteries has an important effect on the useful life of rechargeable batteries. Simulation results can be used to better engineer batteries to extend their useful life for various applications.

Aging Mechanisms

Rechargeable lithium-ion batteries are widely used in a variety of applications, including portable electronic devices and electric vehicles. The performance of a battery is affected strongly by repeated charging and discharging cycles, which can cause the degradation of the battery capacity over time. The porous electrode theory (*Newman et al., 2004*) is a commonly accepted framework for modeling the charge-discharge behavior of lithium-ion cells. The method is based on a homogenized Newman-type approach that does not consider the details of the pore geometry. The porous electrode theory is based on a concurrent solution of a highly coupled multiphysics-multiscale formulation. For more information, see *Coupled Thermal-Electrochemical Analysis*.

Abaqus/Standard can model the following primary battery aging mechanisms:

- Formation and growth of the Solid Electrolyte Interface (SEI) due to electrolyte decomposition and deposition on the electrode surface.
- SEI growth in cracks at the particle surface.
- Decrease in porosity and active surface area due to SEI growth.
- Metallic lithium plating on the particle surface.
- Dissolution of the cathode.

The first four of the above mechanisms happen at the anode, while the last degradation mechanism happens at the cathode, as shown in *Figure 1*

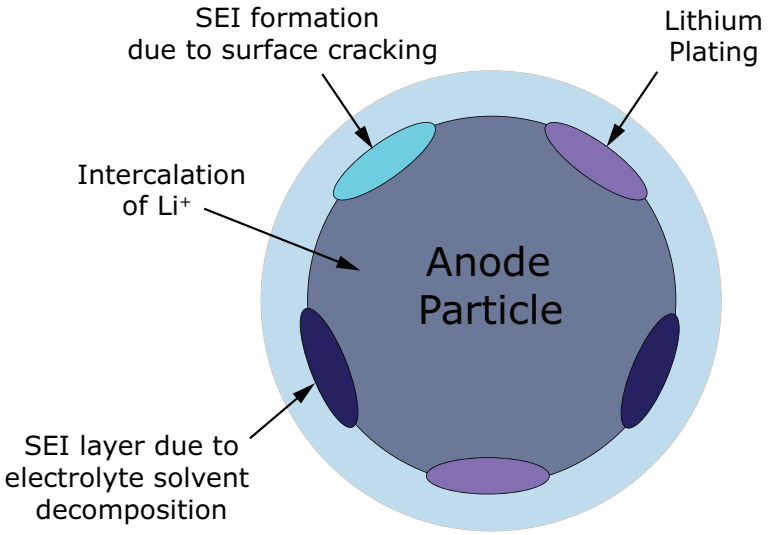


Figure 1: Aging mechanisms in lithium ion battery.

Governing Equations

In addition to intercalation, the chemical reactions corresponding to SEI formation, lithium plating, and SEI growth in cracks contribute to the total current, I_{Tot}, at the particle surface:

$$I_{Tot} = I_{BV} + I_{SEI} + I_{LPL} + I_{Crack}.$$

In the above equation, the expression for the Butler-Volmer current, I_{BV}, is the same as described in *Coupled Thermal-Electrochemical Analysis*. The quantities I_{SEI}, I_{LPL}, and I_{Crack} represent the currents associated with different aging mechanisms; that is, the SEI growth at the particle-electrolyte interfaces, lithium plating, and the SEI growth in the particle surface cracks, respectively. Therefore, aging affects the governing equations of the Newman framework through a modification of the Butler-Volmer intercalation current at the interface between the active particles and the electrolyte. The total current contributes as source terms for the equations that govern the macroscale solid electric potential, fluid-electric potential, and ion concentration fields (see *Coupled Thermal-Electrochemical Analysis*) and, therefore, affects the overall performance of the battery over time.

In the absence of any aging effects, $I_{Tot} = I_{BV}$, and the other currents are zero. However, with aging, the combined effects of the other currents result in a progressive reduction in the magnitude of I_{Tot} and an associated loss in capacity.

The combination of SEI formation and lithium plating also results in the growth of a surface film on the solid particles of the anode, which reduces the effective surface area between the particles and the electrolyte. The discussions that follow outline the constitutive equations for the currents I_{SEI}, I_{LPL}, and I_{Crack} and describe the evolution of the surface film at the surface of the active electrode particles.

Solid Electrolyte Interface

The basic form of the Butler-Volmer kinetics is used to characterize the current at the interface of the particle and the electrolyte, where the SEI layer forms and grows progressively. The expression for I_{SEI} is as follows:

$$I_{SEI} = I_{0,SEI}\left\{\exp\left[\frac{\alpha_{a,SEI}\, z F \eta_{SEI}}{R(\theta-\theta^z)}\right] - \exp\left[\frac{-\alpha_{c,SEI}\, z F \eta_{SEI}}{R(\theta-\theta^z)}\right]\right\},$$

$$I_{0,SEI} = F k_{0,SEI} C_{EC}^{s},$$

$$\eta_{SEI} = \Phi_s - \Phi_e - E_{SEI} - R_{E,SEI} I_{Tot},$$

$$R_{E,SEI} = \frac{\left(\delta_{film} + \delta^0_{film}\right)}{\sigma_{SEI}},$$

where $I_{0,SEI}$ and η_{SEI} are the exchange current density and equilibrium potential of SEI formation, respectively, and $R_{E,SEI}$ is the electrical resistance at the solid electrolye interface.

In the above equations,

F
: is Faraday's constant;

R
: is the gas constant;

z
: is the charge number of the lithium ion battery;

θ
: is temperature;

θ^z
: is the absolute zero temperature;

$K_{0,SEI}$
: is the kinetic rate constant of SEI;

$\alpha_{c,SEI}, \alpha_{a,SEI}$
: are the cathodic and anodic transfer coefficients, respectively;

E_{SEI}
: is the open circuit potential (OCP) as a function of θ;

$\delta^0_{film}, \delta_{film}$
: are the initial thickness and evolving thickness of the SEI film layer;

σ_{SEI}
: is the electrical conductivity of the SEI film layer;

ϕ_s, ϕ_e
: are the electric potentials in the solid and electrolyte phases, respectively; and

C^s_{EC}
: is the concentration of Ethylene Carbonate (EC) on the anode particle surface.

The surface concentration, C^s_{EC}, evolves during the analysis based on the bulk concentration of ethylene carbonate in the electrolyte,

C^{bulk}_{EC} ,as:

$$-D_{EC}\frac{C_{EC}^{s}-C_{EC}^{bulk}}{\delta_{film}}=\frac{-I_{SEI}}{F},$$

where D_{EC} is the diffusivity of EC and δ_{film} is the thickness of the SEI film layer on the particle. The computation of δ_{film} is explained in *Surface Film Growth*.

SEI Reformation

During the regular charging-discharging of a lithium battery, the graphite particle at the anode can expand or contract based on lithiation or delithiation of ions. This change in volume can cause cracking to occur on the particle and SEI layer. The cracked surfaces are then exposed to the electrolyte, which leads to the formation of a new SEI layer. The effects of this additional SEI layer enter the formulation through the current I_{Crack} defined as:

$$I_{SEI_reform}=-I_{0,SEI_reform}f_{crack}\left\{\exp\left[\frac{-zF\eta_{Crack}}{R(\theta-\theta^{z})}\right]\right\},$$

$$\eta_{Crack}=\Phi_{s}-\Phi_{e}-E_{SEI},$$

where I_{0,SEI_reform} and η_{Crack} are the exchange current density of the SEI reformation and the equilibrium potential of the SEI reformation, respectively. Typically, I_{0,SEI_reform} = $I_{0,SEI}$.

In the above equation, f_{Crack} is a cracking function dependent on the intercalation in the particle.

Lithium Plating

Lithium plating occurs only when the electric potential at the anode is negative; that is, $\phi_{s}-\phi_{e}<0$. This process is assumed to be irreversible, which means that stripping of lithium in the subsequent discharge is neglected. The lithium plating contribution to the total current is I_{LPL} and is given as:

$$I_{LPL}=-I_{0,LPL}\left\{\exp\left[\frac{-\alpha_{c,LPL}zF\eta_{LPL}}{R(\theta-\theta^{z})}\right]\right\},$$

$$\eta_{LPL}=\Phi_{s}-\Phi_{e}-R_{E,SEI}I_{Tot},$$

where $I_{0,LPL}$ and η_{LPL} are the exchange current density and the equilibrium potential of lithium plating, respectively, and $R_{E,SEI}$ is the electrical resistance at the solid electrolye interface.

In the above equations, $\alpha_{c,LPL}$ is the cathodic transfer coefficient.

Lithium plating typically dominates a narrow portion of the anode near the separator after a certain number of cycles when the potential is negative. The lithium plating is associated with a drop of anode porosity associated with SEI film growth, which can increase the local electrolyte potential gradient in the anode. After hundreds of cycles of cyclic charging, there can be an exponential increase of lithium plating, as well as local pore clogging near the anode/separator interface.

Surface Film Growth

SEI and lithium metal together form the surface film covering the active particles at the anode. Within the film, the time evolution of the concentrations of SEI and plated lithium are governed by the following equations:

$$\frac{\partial C_{SEI}}{\partial t}=-\frac{a_{s}(I_{SEI}+I_{Crack})}{zF}-\frac{a_{s}I_{LPL}}{zF}\beta,$$

$$\frac{\partial C_{LPL}}{\partial t}=-\frac{a_{s}I_{LPL}}{F}\left(1-\beta\right),$$

where C_{SEI}, C_{LPL} are the molar concentrations of the SEI and plated lithium, respectively, per unit volume of the electrode; a_s is the wetted particle surface area per unit volume; and β is the fraction of plated lithium that can convert to the SEI.

The surface film is characterized by an equivalent thickness, δ_{film}, which is defined as the ratio of the total volume of the SEI and lithium metal to the specific surface area per unit volume of electrode, a_s. The equivalent thickness of the surface film is determined by the concentrations of the SEI and plated lithium as:

$$\delta_{film} = \frac{1}{a_s}\left(\frac{C_{SEI}M_{SEI}}{\rho_{SEI}} + \frac{C_{LPL}M_{LPL}}{\rho_{LPL}}\right),$$

where M_{SEI}, M_{LPL} are the molar weights of the SEI and plated lithium, and ρ_{SEI}, ρ_{LPL} are the molar densities of the SEI and plated lithium.

Using the computed film thickness, one can compute the ionic resistance, R_{SEI}, and electrical resistance, $R_{E,SEI}$, of the SEI layer. Since this model neglects the stripping of lithium metal, the plated lithium is considered isolated from the main electron-conduction matrix. Therefore, the ionic resistance is determined only by SEI as:

$$R_{SEI} = \omega_{SEI}\frac{\delta_{film}}{\kappa_{SEI}},$$

where ω_{SEI} is the volume fraction of the SEI in the film, and κ_{SEI} is the ionic conductivity of the SEI layer.

The computed R_{SEI} is used in the Butler-Volmer kinetics as described in *Coupled Thermal-Electrochemical Analysis*.

Cathode Dissolution

The dissolution of cathode happens above a certain prescribed voltage where, in the case of lithium ion batteries, the hydrofluoric acid starts to dissolve the active cathode material, resulting in a reduction of active surface area and solid volume fraction of the cathode. The dissolution mechanism is governed by the following equations:

$$I_{Diss} = I_{0,Diss}\left\{\exp\left[\frac{F\eta_{Diss}}{R(\theta-\theta^z)}\right]\right\},$$

$$\eta_{Diss} = \Phi_s - \Phi_e - E_{Diss},$$

where $I_{0,Diss}$ and η_{Diss} are the exchange current density and the equilibrium potential of the cathode dissolution, respectively. E_{Diss} is the potential above which the dissolution of active material happens. The cathode dissolution effect is captured as a change in solid volume fraction, ϵ_s, that affects the wetted surface area, a_s, and, therefore, further reduces the capacity of the battery.

$$\epsilon_{cathode} = \epsilon^0_{cathode} - \frac{I_{Diss}\Delta t}{C^{max}_{s,P} l_{cathode} F},$$

where $l_{cathode}$ is the length of the cathode.

Defining the Properties for the Aging Mechanisms

You must specify the constants required to define each of the aging mechanisms.

Solid Electrolyte Interface Properties

You must define tables of properties required to activate the computation of the SEI current.

Input File Usage: Use the following options to define the properties to model aging due to the solid electrolyte interface growth:

*PARAMETER TABLE, TYPE="ABQ_EChemPET_Electrode_Particle_SEI"

$\alpha_{a,SEI}$, $\alpha_{c,SEI}$, $K_{0,SEI}$, C_{EC}^{bulk}, $C_{EC,0}^{s}$, $C_{0,SEI}$, β, ρ_{SEI}, M_{SEI}, z

Use the following options to specify a tabular form of $I_{0,SEI}$:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_SEI_Tabular"

$I_{0,SEI}$, C_{ec}^{S}

Use the following options to define the open circuit potential for a layer in tabular format:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_SEI_OCPTabular"

E_{SEI}, C_{ec}^{S}

Use the following options to specify a tabular form of D_{EC}:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_SEI_DiffTabular"

D_{EC}, C_{ec}^{S}

Use the following options to specify a tabular form of ionic conductivity, κ_{SEI}:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_SEI_IonCondTabular"

κ_{SEI}, C_{ec}^{S}

Use the following options to specify a tabular form of electric conductivity, σ_{SEI}:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_SEI_ElecCondTabular"

σ_{SEI}, C_{ec}^{S}

SEI Reformation Properties

You must define tables of properties required to activate the SEI and to compute the SEI reformation after a crack on the particle .

Input File Usage: Use the following options to define the properties for reformation of the SEI:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_CrackSEI_Tabular"

I_{0,SEI_reform}

C_{EC}^{S}

Use the following options to define the cracking function in tabular format:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_SEI_CrackFunction_Tabular"

f_{crack}, $\widehat{C}_{s,P}$

Lithium Plating Properties

You must define tables of properties required to activate the computation of the lithium plating current.

Input File Usage: Use the following options to define the properties for lithium plating:

*PARAMETER TABLE, TYPE="ABQ_EChemPET_Electrode_Particle_LPL"

$\alpha_{c,LPL}$,C^0_{LPL},ρ_{LPL},M_{LPL}

Use the following options to specify a tabular form of $I_{0,LPL}$:

*TABLE COLLECTION
*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_LPL_Tabular"

$I_{0,LPL}$, C_{LPL}

Cathode Dissolution Properties

You must define tables of properties required to activate the computation of the cathode dissolution.

Input File Usage: Use the following options to define the properties for cathode dissolution:

*PARAMETER TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_ElectrodeDissolution"

$l_{cathode}$

Use the following options to specify a tabular form of $I_{0,Diss}$ as a function of temperature:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_ElecDiss_Tabular"

$I_{0,Diss}$, θ

Use the following options to define the open circuit potential for a layer in tabular format:

*PROPERTY TABLE,
TYPE="ABQ_EChemPET_Electrode_Particle_ElecDiss_OCPTabular"

E_{diss}, θ

User-Defined Custom Currents

You can define your own custom battery aging mechanisms using the following user subroutines:

- *UECHEMECDSEI* to define a customized SEI aging mechanism,
- *UECHEMECDSEIREFORM* to define a customized SEI reformation aging mechanism,

- *UECHEMECDLPL* to define a customized lithium plating aging mechanism, and
- *UECHEMECDDISS* to define a customized electrode dissolution aging mechanism.

Input File Usage: Use the following options to activate the definition of a customized SEI aging mechanism using user subroutine *UECHEMECDSEI* :

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_ISEIUser*"
1

Use the following options to activate the definition of a customized SEI reformation aging mechanism using user subroutine *UECHEMECDSEIREFORM* :

**PARAMETER TABLE*,
TYPE="*ABQ_EChemPET_Electrode_Particle_ISEIReformUser*"
1

Use the following options to activate the definition of a customized lithium plating aging mechanism using user subroutine *UECHEMECDLPL*:

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_ILPLUser*"
1

Use the following options to activate the definition of a customized electrode dissolution aging mechanism using user subroutine *UECHEMECDDISS* :

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_IDissUser*"
1

Defining and Controlling an Aging Analysis

A battery analysis typically consists of four (or more) steps in the following order:

- A charging step, in which the battery is subjected to a constant charging current. The voltage is monitored, and the step is terminated when the voltage increases to a specified cut-off value.
- A constant voltage step, in which the battery is subjected to a constant prescribed voltage. The reaction current is monitored, and the step is terminated when the reaction current drops below a specified threshold value.
- A resting step, in which the battery is not subjected to any prescribed current or voltage.
- A discharge step, in which a constant current is drawn from the battery. The voltage is monitored, and the step is terminated when the voltage drops to a specified cut-off value.

You can use the step control mechanism (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) together with sensors (see *Defining Sensors*) to model the workflow described above. In particular, you can associate the history output of the voltage or the current at specific locations in the model with sensor definitions, and terminate a step when the sensor reaches a specified cut-off value or threshold .

For an aging analysis, these four steps are repeated 100s to 1000s of times to capture the full degradation and aging that happens during the normal operation of a battery.

Output

In addition to the output variables available for the coupled thermal-electric procedures, the coupled thermal-electrochemical procedures, the coupled thermal-electrochemical-structural procedures, and the thermal-electrochemical-structural-pore pressure procedures, you can request the following element integration point output variables related to aging in Abaqus/Standard:

CONCSEI_*i*	SEI concentration in the film for particle *i*.
CONCLPL_*i*	Plated lithium concentration in the film for particle *i*.
CONCSOL_*i*	Concentration of solute on the surface of particle *i*.
ECDSEI_*i*	SEI Butler-Volmer current for particle *i*.
ECDLPL_*i*	Butler-Volmer current for plated lithium in particle *i*.
ECDCRACKSEI_*i*	Butler-Volmer current for particle i due to reformation of the SEI.
ECDELECDISS_*i*	Butler-Volmer current for particle *i* due to dissolution.
ECDTOTAL_*i*	Total current due to the different aging mechanisms.
OVERPOTSEI_*i*	SEI overpotential in particle ***i***.
OVERPOTLPL_*i*	LPL overpotential in particle ***i***.
OVERPOTDISS_*i*	Overpotential in particle ***i*** due to electrode dissolution.
OVERPOTCRACKSEI_*i*	Overpotential in particle ***i*** due to SEI cracking and reformation.
SEILPLTHICK_*i*	Thickness of the combined SEI and LPL layer in particle ***i***.
SOC_*i*	State of charge in particle ***i***.

References

- Yang, X-G., Y. Leng, G. Zhang, S. Ge, and C-Y. Wang, “Modeling of Lithium Plating Induced Aging of Lithium-Ion Batteries: Transition from Linear to Nonlinear Aging,” *Journal of Power Sources*, vol. 360, no. 28-40, 2017.
- Kindermann, F., K. Jonas, A. Frank, and A. Jossen, “A SEI Modeling Approach Distinguishing between Capacity and Power Fade,” *Journal of the Electrochemical Society*, vol. 164, no. 12, 2017.
- Newman, J., , and K. E. Thomas-Alyea, Electrochemical Systems, Wiley-Interscience , Third Edition , 2004.

Modeling Thermal Runaway in Batteries

Products: Abaqus/Standard

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*

Overview

Abaqus/Standard provides a set of capabilities to model thermal runaway in batteries.

Thermal runaway of lithium ion batteries is a very rapid, self-feeding, uncontrolled event that releases large amounts of heat due to the exothermic reactions of the various battery components, such as the electrolyte, SEI layer, and decomposition of the lithium at the electrodes. It has an important effect on the usage and safety of rechargeable batteries. Simulation results can be used to engineer better batteries that remain safe during different operating conditions for various applications.

Thermal Runaway Mechanisms

Rechargeable lithium-ion batteries are widely used in a variety of applications, including portable electronic devices and electric vehicles. The safety of a battery under different operating conditions plays an important role in its adoption and widespread usage. In a battery pack, the heat generated due to thermal runaway within a single cell can result in a chain reaction that can trigger other cells to undergo thermal runaway as well, leading to unsafe conditions such as explosions and fires in electric vehicles or cell phones. Different factors can influence thermal runaway; for example, overcharging, high ambient temperature, and battery age.

The porous electrode theory (*Newman et al., 2004*) is a commonly accepted framework for modeling the charge-discharge behavior of lithium-ion cells. The method is based on a homogenized Newman-type approach that does not consider the details of the pore geometry. Porous electrode theory is based on a concurrent solution of a highly coupled multiphysics-multiscale formulation. For more information, see *Coupled Thermal-Electrochemical Analysis*.

Thermal runaway is a rapid event, with thermal effects and decomposition of the various species in the battery being the dominating factors. During a thermal runaway simulation, the porous electrode theory framework is disabled in the regions where thermal runaway is active. The evolution and decomposition of the ion concentration and the lithium concentrations in the particles are treated as locally adiabatic events, where the effects of diffusion and convection on these fields are ignored. Consequently, the ion concentration field and the electric potential field in the fluid phase are locally deactivated and are not sent to the solver.

Abaqus/Standard can model the following thermal runaway mechanisms:

- Decomposition of the lithium concentration in the electrodes.
- Decomposition of the SEI.
- Decomposition of the electrolyte.
- Decomposition of the state of charge (SOC)

The SEI decomposition occurs at the anode only. The decomposition of lithium concentration in the electrodes and the SOC happen at both the anode and cathode.

Governing Equations

The total heat, Q_{TOT}, generated due to the various exothermic reactions from the decomposition of the different constituents in a battery is:

$$Q_{TOT} = Q_{CE} + Q_{SOC} + Q_{SPC} + Q_{SEI}.$$

In the above equation, the total heat generated is assumed to be the sum of the heat generated from decomposition of the electrolyte (Q_{CE}), decomposition of the SOC (Q_{SOC}), decomposition of the particle species (Q_{SPC}), and decomposition of the SEI (Q_{SEI}).

Electrolyte Heat Generation and Electrolyte Concentration Evolution

The rate of heat generated due to the electrolyte decomposition is:

$$\frac{dQ_{CE}}{dt} = -\eta_e \rho_e h_e \epsilon_e \frac{dC_{ne}}{dt},$$

The above rate of heat generation depends on the rate at which C_e evolves locally:

$$\frac{dC_e}{dt} = A_e C_e \exp\left[-\frac{E_e}{R(\theta-\theta^z)}\right].$$

In the above equations,

R
: is the gas constant;

θ
: is the temperature;

θ^z
: is the absolute zero temperature;

A_e
: is the frequency factor for electrolyte decomposition in units of $\frac{1}{T}$;

E_e
: is the activation energy for electrolyte decomposition;

h_e
: is the enthalpy of electrolyte decomposition;

C_{ne}
: is the normalized ion concentration in the electrolyte. The normalization is with respect to C_e^{ref} (see *Coupled Thermal-Electrochemical Analysis*);

ρ_e
: is the electrolyte density;

ϵ_e
is the volume fraction of the electrolyte; and

η_e
is the fraction of heat generated due to electrolyte decomposition that contributes toward the total heat generated.

SOC Heat Generation and SOC Evolution

The rate of heat generated due to the SOC decomposition for a particle is:

$$\frac{dQ_{SOC}}{dt} = -\eta_{soc}\epsilon_s\epsilon_{sParticle}\rho_{soc}h_{soc}\frac{dSOC}{dt},$$

The above rate of heat generation depends on the rate at which SOC evolves locally:

$$\frac{dSOC}{dt} = A_{soc}SOC\ \exp\left[-\frac{E_{soc}}{R(\theta-\theta^z)}\right].$$

In the above equations,

A_{soc}
is the frequency factor for SOC decomposition in units of $\frac{1}{T}$;

E_{soc}
is the activation energy for SOC decomposition;

h_{soc}
is the enthalpy of SOC decomposition;

SOC
is the state of charge (SOC);

ρ_{soc}
is the density that is associated with SOC for heat computation;

$\epsilon_{sParticle}$
is the volume fraction of the microscale particle with respect to the solid volume fraction of the electrode (see *ABQ_EChemPET_Electrode_Particles*);

ϵ_s
is the volume fraction of the solid phase in the electrode; and

η_{soc}
is the fraction of heat generated due to SOC decomposition that contributes toward the total heat generated.

Lithium Species Particle Concentration: Heat Generation and Evolution at the Cathode

The rate of heat generated due to the decomposition of the lithium species in the cathode particle is:

$$\frac{dQ_{SPC}}{dt} = -\eta_{spc}\epsilon_s\epsilon_{sParticle}\rho_{spc}h_{spc}\frac{dC_s}{dt},$$

The above rate of heat generation depends on the rate at which C_s evolves locally:

$$\frac{dC_s}{dt} = A_{spc}\frac{C_s}{C_{s,P}^{\max}}\left(1-\frac{C_s}{C_{s,P}^{\max}}\right)\exp\left[-\frac{E_{spc}}{R(\theta-\theta^z)}\right].$$

In the above equations,

A_{spc}
is the frequency factor for particle species decomposition in units of $\frac{1}{T}$;

E_{spc}
is the activation energy for particle species decomposition;

h_{spc}
is the enthalpy of species decomposition;

$\frac{C_s}{C_{s,P}^{\max}}$
is the particle species concentration normalized with respect to the theoretical max particle species concentration (see *ABQ_EChemPET_Electrode_Particle_Layer_Diffusion*);

ρ_{spc}
is the density that is associated with the particle species for heat computation; and

η_{spc}
is the fraction of heat generated due to particle species decomposition that contributes toward the total heat generated.

Lithium Species Particle Concentration: Heat Generation and Evolution at the Anode

The rate of heat generated due to the decomposition of the lithium species in the cathode particle is:

$$\frac{dQ_{SPC}}{dt} = -\eta_{spc}\epsilon_s\epsilon_{sParticle}\rho_{spc}h_{spc}\frac{dC_s}{dt},$$

The above rate of heat generation depends on the rate at which C_s evolves locally:

$$\frac{dC_s}{dt} = A_{spc}\frac{C_s}{C_{s,P}^{\max}}\exp\left[-\frac{E_{spc}}{R(\theta-\theta^z)}\right]\exp\left[-\frac{z}{z_0}\right].$$

In the above equations,

A_{spc}
is the frequency factor for particle species decomposition in units of $\frac{1}{T}$;

E_{spc}
is the activation energy for particle species decomposition;

h_{spc}
is the enthalpy of species decomposition;

$\frac{C_s}{C_{s,P}^{\max}}$
is the particle species concentration normalized with respect to the theoretical max particle species concentration (see *ABQ_EChemPET_Electrode_Particle_Layer_Diffusion*);

ρ_{spc}
is the density that is associated with the particle species for heat computation;

η_{spc}
is the fraction of heat generated due to particle species decomposition that contributes toward the total heat generated;

z
the SEI thickness, is added to the anode decomposition to include tunneling effects (*Richard et al., 1999*); and

z_0
is the SEI thickness at the start of thermal runaway.

SEI: Heat Generation and Evolution at the Anode

The rate of heat generated due to the decomposition of the SEI layer is:

$$\frac{dQ_{SEI}}{dt} = -\eta_{sei}\epsilon_s\epsilon_{sParticle}\rho_{sei}h_{sei}\frac{dC_{sei}}{dt},$$

The above rate of heat generation depends on the rate at which C_{SEI} evolves locally:

$$\frac{dC_{sei}}{dt} = A_{sei}C_{sei}\ \exp\left[-\frac{E_{sei}}{R(\theta-\theta^z)}\right].$$

In the above equations,

A_{sei}
is the frequency factor for SEI decomposition in units of $\frac{1}{T}$;

E_{sei}
is the activation energy for particle SEI decomposition;

C_{SEI}
is the concentration of lithium in the SEI;

h_{sei}
is the enthalpy of SEI decomposition;

ρ_{sei}
is the density that is associated with the particle SEI for heat computation; and

η_{sei}
is the fraction of heat generated due to particle SEI decomposition that contributes toward the total heat generated.

Defining the Properties for the Thermal Runaway Mechanisms

You must specify the material parameter, thermal runaway temperature above which the various species can decompose and release heat.

Input File Usage: Use the following options:

**MATERIAL*
**POROUS ELECTRODE THEORY*
**THERMAL RUNAWAY*, TEMPERATURE=*value*

Properties for Heat Generation due to Particle Species Decomposition

You must define tables of properties required to activate the computation of the heat generated due to the decomposition of lithium in particles.

Input File Usage: Use the following options to define the properties to model particle species decomposition during thermal runaway:

**MATERIAL*
**PARAMETER TABLE*,
TYPE="*ABQ_EChemPET_Electrode_Particle_TR_SpeciesConc*"

A_{spc}, E_{spc}, h_{spc}, ρ_{spc}, z_0,η_{spc}

Properties for Heat Generation due to SEI Decomposition

You must define tables of properties required to activate the heat generation due to SEI decomposition, which is typically present only on the anode side.

Input File Usage: Use the following options to model the thermal runaway of SEI:

**MATERIAL*

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_TR_SEI*"

A_{sei}, E_{sei}, h_{sei}, ρ_{sei}, SEI_{init}, η_{sei}

Properties for Heat Generation due to SOC Decomposition

You must define tables of properties required to activate the heat generation due to SOC decomposition.

Input File Usage: Use the following options to model the thermal runaway of SOC:

**MATERIAL*

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrode_Particle_TR_SOC*"

A_{soc}, E_{soc}, h_{soc}, ρ_{soc}, η_{soc}

Properties for Heat Generation due to Electrolyte Decomposition

You must define tables of properties required to activate the heat generation due to electrolyte decomposition.

Input File Usage: Use the following options to model the thermal runaway of electrolyte:

**MATERIAL*

**PARAMETER TABLE*, TYPE="*ABQ_EChemPET_Electrolyte_TR*"
A_e, E_e, h_e, ρ_e, η_e

Defining and Controlling a Thermal Runaway Analysis

Thermal runaway is a rapid, short-term event. In Abaqus/Standard, this event is modeled as an adiabatic process, where the electrochemical fields (such as ion concentration, electrolyte potential in the fluid, and microscale particle concentrations) are evolved locally as state variables using the equations described in the *Governing Equations*. Abaqus/Standard solves for only the following fields:

- temperature,
- displacement, and
- solid electric potential.

You must activate thermal runaway in a step. If you do not activate it, the decomposition of the various species is not accounted for even if the local temperature is above the thermal runaway trigger temperature. You can run a thermal runaway scenario in a multi-step battery analysis. The first set of steps can be the charge-constant voltage hold-discharge steps, and you can define sensors to end a step when the temperature is above the specified thermal runaway trigger temperature. The subsequent step can be a thermal runaway step, where you can model the heat generated, temperature rise, and evolution of the different species. You do not need to define all the thermal runaway parameters. You can selectively define thermal runaway material parameters that are of interest in the simulation.

During a thermal runaway step, you might observe Abaqus/Standard taking small time steps to resolve the temporal evolution associated with a large rise in temperature due to the large amounts of heat being generated. You can use step controls (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) together with sensors (see *Defining Sensors*) to model the workflow described above. In particular, you can associate the history output of the temperature at specific locations in the model with sensor definitions to terminate a step when a sensor reaches a predefined cut-off value or threshold.

Input File Usage: Use the following options to activate thermal runaway in a step:

**COUPLED THERMAL-ELECTROCHEMICAL*, DELTMX=*value*, THERMAL RUNAWAY
**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, DELTMX=*value*, THERMAL RUNAWAY

Output

In addition to the output variables available for the coupled thermal-electrical procedures, you can request the following element integration point output variables related to thermal runaway in Abaqus/Standard:

QTRLI_*i*	Heat generated due to decomposition of lithium for particle *i*.
QTRSEI_*i*	Heat generated from SEI decomposition for particle *i*.
QTRSOC_*i*	Heat generated from SOC decomposition for particle *i*.
QTR_CE	Heat generated from decomposition of the electrolyte.

References

- Newman, J., and K. E. Thomas-Alyea, "Electrochemical Systems," Wiley-Interscience, Third Edition, 2004.
- Coman, P. T., E. C. Darcy, C. T. Veje, and R. E. White, "Numerical Analysis of Heat Propagation in a Battery Pack using a Novel Technology for Triggering Thermal Runaway," *Applied Energy*, vol. 203, no. 189-200, 2017.
- Richard, M. N., and J. R. Dahn, "Accelerating Rate Calorimetry Study on the Thermal Stability of Lithium Intercalated Graphite in Electrolyte, I. Experimental," *Journal of the Electrochemical Society*, vol. 146 (6), no. 2067-2077, 1999.

Modeling Solid Electrolytes and Solid-State Batteries

Products: Abaqus/Standard

References:

- *Parameter Table Type Reference*
- *Property Table Type Reference*
- *Modeling a solid electrolyte and solid-state battery*

Overview

Unlike a regular Li-ion battery that uses a liquid electrolyte and porous electrodes, a solid-state battery uses a solid electrolyte, which also acts as a separator. A solid-state battery has a higher energy density and is safer than a lithium ion battery.

Solid-State Batteries

Solid-state batteries are composed of a solid anode, a solid cathode, and a solid electrolyte. The electrolyte also acts as a separator as shown in *Figure 1*.

Solid-state lithium ion batteries are manufactured using thin film methods, with the thicknesses of different regions in the order of a few micrometers. Since the electrodes and electrolyte are solid, all the electrochemical reactions happen at the anode-electrolyte and cathode-electrolyte interfaces. In this section, for consistency, the electrolyte region can be interchangeably referred to as the separator. The anode is typically made of metallic lithium, while the cathode is based on Lithium Cobalt Oxide ($LiCoO_2$). The electrolyte is solid and might consist of an amorphous Lithium Phosphate (Li_3PO_4), which can be N-doped and commonly referred to as LiPON. Current collectors might be placed on the outer sides of each electrode.

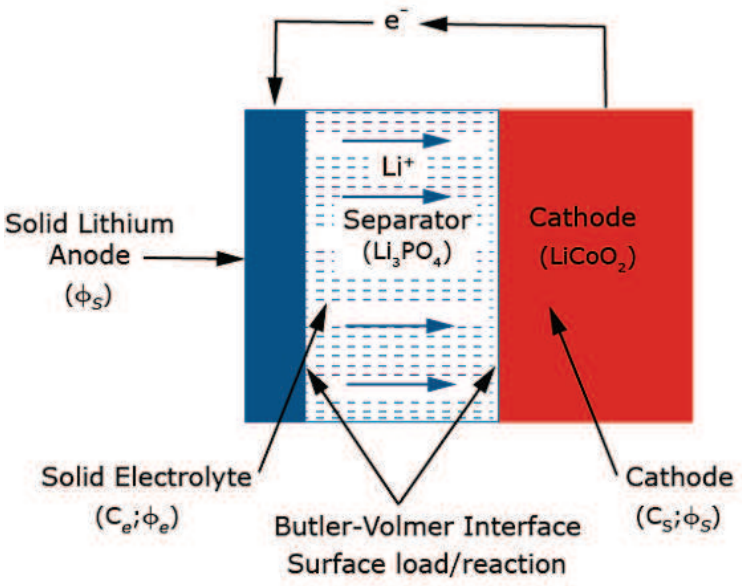


Figure 1: Schematic diagram of a solid electrolyte battery.

During a discharge process, a reversible electrochemical reaction at the anode-electrolyte interface converts Li to Li^+:

$$\mathrm{Li} - \Theta \leftrightarrow \mathrm{Li}^+ + e^- + \Theta.$$

Here, $\mathrm{Li} - \Theta$ represents lithium in the solid electrode, which through the deintercalation reaction, transforms into a lithium ion in the electrolyte, Li^+, plus an electron, e^-, and leaves behind a vacancy (or intercalation site) in the solid, Θ. The reverse reaction corresponds to the intercalation of lithium into the solid electrode. In the solid electrolyte, only a fraction of Li^+ ions are mobile and denoted by a fraction δ. It is assumed that a portion of the mobile lithium ions would kinetically bond with the solid electrolyte and become immobile, which can be expressed as:

$$Li^+ + e^- \underset{k_d}{\overset{k_r}{\rightleftharpoons}} Li_0,$$

where Li_0 refers to the immobile Li bonded with anions in the solid electrolyte and k_r, k_d are the reverse reaction rate and dissociation rate, respectively. The net production rate of Li^+ ions that can pass through the solid electrolyte is obtained as:

$$r = k_r\left[\left(C_0\delta\right)^2 - C_{Li^+}^2\right],$$

where C_0 is the total amount of lithium. The transport of lithium ion in the solid electrolyte is driven by both diffusion and migration.

At the solid electrolyte-cathode interface, the charge transfer reaction can be expressed as:

$$LiCoO_2 \rightleftharpoons xLi^+ + xe^- + Li_{(1-x)}CoO_2$$

where x is the stoichiometric coefficient. After the charge transfer reaction at the interface, the lithium intercalates within the cathode through a diffusion process.

Terminology

In the discussions that follow, "species" refers to the chemical species that participates in the primary electrochemical reactions in the battery. It is assumed in this section that the solid electrolyte battery is lithium-based. Therefore, "species" refers to lithium; but, in principle, it could be another chemical species.

An important physical mechanism that is active at the cathode of a solid electrolyte battery is diffusion of the chemical species. The discussion and user interface for the section on cathode behavior refer to "species." However, "species" can be interchanged with lithium for the particular kind of solid electrolyte battery discussed here.

Governing Equations

The following wide range of physical phenomena determine the operation of a solid state battery:

- Conduction of electrons in the solid electrodes.
- Butler-Volmer charge transfer reactions at the interface of the anode-solid electrolyte.
- Diffusion and migration of ions in the solid electrolyte.
- Butler-Volmer charge transfer reactions at the interface of the solid electrolyte and the cathode.
- Diffusion of species (lithium) in the solid cathode.
- Heat generation due to current flow.

Conduction of Electrons in the Solid Electrodes

The equilibrium equation for the current density, $\mathbf{I}_s$, in the solid phase of the anode and cathode is given by:

$$\nabla \cdot \mathbf{I}_s = 0,$$

where

$$\mathbf{I}_s = -\mathbf{K}_s^{\text{eff}} \cdot \nabla \Phi_s.$$

Here, $\mathbf{K}_s^{\text{eff}}$ is the effective electrical conductivity of the solid phase, and Φ_s is the electric potential in the solid phase.

Interaction at the Anode-Solid Electrolyte Interface

At the interface between the separator and the solid lithium anode, a charge-transfer reaction occurs following a Butler-Volmer kinetics that is expressed using the following equations:

$$\mathrm{Li} - \Theta \leftrightarrow \mathrm{Li}^+ + e^- + \Theta.$$

Here, $\mathrm{Li} - \Theta$ represents lithium in the solid electrode that, through the deintercalation reaction, transforms into a lithium ion in the electrolyte, Li^+, plus an electron, e^-, and leaves behind a vacancy (or intercalation site) in the solid, Θ.

The general form of the Butler-Volmer current expression at the interface of a solid anode and solid electrolyte is:

$$I_{BV}^{a-s} = I_0^{a-s}\left(\exp\left[\frac{\alpha_a(zF\eta)}{R(\theta-\theta^z)}\right] - \exp\left[\frac{-\alpha_c(zF\eta)}{R(\theta-\theta^z)}\right]\right)$$

and

$$I_0^{a-s} = Fk_a\left(\frac{C_e}{C_e^{ref}}\right)^{\alpha_a},$$

where the overpotential, η, is defined as:

$$\eta = \Phi_s - \Phi_e - E_{\text{ref}}(\theta).$$

In the above equations,

F
: is Faraday's constant;

R
: is the universal gas constant;

z
: is the charge number of the lithium ion battery;

θ
: is temperature;

θ^z
: is the absolute zero temperature;

α_c, α_a
: are the cathodic and anodic transfer coefficients, respectively;

C_e^{ref}
: is the reference value of lithium ion concentration in the separator;

C_e
: is the concentration of mobile lithium ion;

E_{ref}
: is the open circuit potential (OCP) as a function of θ (The typical value in a solid anode is zero.);

ϕ_s
: is the electric potential from the anode side; and

ϕ_e
: is the electric potential from the separator side.

The electrochemical interaction at the solid electrode-separator interface is represented on the solid electrode side as a surface current density of magnitude $-S_{fac}I_{BV}^{a-s}$. The dimensionless "surface factor," S_{fac}, captures the effects of surface irregularities that can result in the availability of a larger surface area over which the electrochemical interaction happens. On the separator side of the interface, the electrochemical interaction results in two separate surface fluxes, both applied on the solid electrolyte phase: an electrolyte current density of magnitude $S_{fac}I_{BV}^{a-s}$, and an ion flux of magnitude $\frac{S_{fac}I_{BV}^{a-s}}{F}$. For more details on the electrochemical governing equation, see *Conduction of Electrons in the Solid Electrodes* and *Diffusion and Migration in the Solid Electrolyte*. The current surface interaction load is treated as surface flux at the interface of the anode and solid electrolyte.

Diffusion and Migration in the Solid Electrolyte

The transport of lithium ions in the solid electrolyte is by a combination of diffusion and migration, which is modeled using the Nernst-Planck equation and Fick's second law:

$$\mathbf{N}_e = -\mathbf{D}_e.\nabla C_e + \left(\frac{F}{R(\theta-\theta^z)}\right)\mathbf{D}_e C_e.\nabla\Phi_e,$$

$$\frac{dCe}{dt} = -\nabla\cdot\mathbf{N}_e,$$

where

$\mathbf{N}_e$
: is the flux of lithium ions;

$\mathbf{D}_e$
: is the diffusion coefficient; and

C_e
: is the concentration of lithium ion.

The equilibrium equation for the current density in the electrolyte, $\mathbf{I}_e$, is given by:

$$\nabla\cdot\mathbf{I}_e = 0,$$

where

$$\mathbf{I}_e = -\mathbf{K}_e^{\text{eff}}\cdot\nabla\Phi_e.$$

Here, $\mathbf{K}_e^{\text{eff}}$ is the effective electrical conductivity of the electrolyte, and Φ_e is the electric potential in the electrolyte phase.

Interaction at the Solid Electrolyte - Cathode Interface

At the solid electrolyte-cathode interface, the charge transfer reaction can be expressed as:

$$LiCoO_2 \rightleftharpoons xLi^+ + xe^- + Li_{(1-x)}CoO_2$$

This charge-transfer reaction occurs following a Butler-Volmer kinetics that is expressed using the following equations:

$$I_{BV}^{s-c} = I_0^{s-c}\left\{\exp\left[\frac{\alpha_c(zF\eta)}{R(\theta-\theta^z)}\right] - \exp\left[\frac{-(1-\alpha_c)(zF\eta)}{R(\theta-\theta^z)}\right]\right\},$$

where the overpotential, η, is defined as:

$$\eta = \Phi_s - \Phi_e - E_{\text{ref}}(\theta),$$

and I_0^{s-c} is

$$I_0^{s-c} = Fk_c\left(\frac{(C_{s,\max}-C_s)C_e}{(C_{s,\max}-C_{s,\min})C_e^{ref}}\right)^{\alpha_c}\left(\frac{C_s-C_{s,\min}}{C_{s,\max}-C_{s,\min}}\right)^{(1-\alpha_c)}$$

In the above equations,

F
: is Faraday's constant;

R
: is the universal gas constant;

z
: is the charge number of the battery;

θ
: is temperature;

θ^z
: is the absolute zero temperature;

α_c
: is the cathodic transfer coefficients, respectively;

C_e^{ref}
: is the reference value of lithium ion concentration in the separator;

C_e
: is the concentration of mobile lithium ion;

E_{ref}
: is the open circuit potential (OCP) as a function of θ;

ϕ_s
: is the electric potential from the cathode side;

ϕ_e
is the electric potential from the separator side;

$C_{s,\max}, C_{s,\min}$
are the maximum and minimum possible concentration of lithium in the cathode; and

C_s
is the concentration of lithium in the cathode.

The electrochemical interaction at the separator-cathode interface is represented on the cathode side as two separate surface fluxes: a surface current density of magnitude $-S_{fac}I_{BV}^{s-c}$ and a species flux of magnitude $\frac{S_{fac}I_{BV}^{s-c}}{F}$. The dimensionless "surface factor," S_{fac}, captures the effects of surface irregularities that can result in the availability of a larger surface area over which the electrochemical interaction happens. On the separator side of the interface, the electrochemical interaction results in two separate surface fluxes, both applied on the solid electrolyte phase: an electrolyte current density of magnitude $S_{fac}I_{BV}^{s-c}$, and an ion flux of magnitude $\frac{S_{fac}I_{BV}^{s-c}}{F}$. For more details on the electrochemical governing equations, see *Conduction of Electrons in the Solid Electrodes*, *Diffusion of Lithium within the Cathode*, and *Diffusion and Migration in the Solid Electrolyte*. The current surface interaction load is treated as surface flux at the interface of the cathode and solid electrolyte.

Diffusion of Lithium within the Cathode

The diffusion of lithium within the cathode is solved using a diffusion equation, which is expressed as:

$$\frac{\partial C_s}{\partial t} = \nabla.(\mathbf{D}_s.(\nabla C_s)),$$

where

D_s
is the diffusion coefficient in the cathode for the species (lithium); and

C_s
is the species (lithium) concentration in the cathode.

Joule Heating at the Electrode–Solid Electrolyte Interface

The total heat generation at the interface of the solid electrode and the solid electrolyte is attributed to two sources: flow of current at the electrode-separator interface and entropy generation. The amount of energy converted into heat from each of the source terms can be scaled using the conversion factors, β_1, β_2:

$$Q_{\text{tot}} = \beta_1 Q_1 + \beta_2 Q_2,$$

where

$Q_1 = S_{fac}I_{\text{BV}}\eta$
is the ohmic loss at the electrode-separator interface; and

$$Q_2 = S_{fac} I_{\text{BV}} \left(\theta - \theta^z\right) \frac{\partial \text{U}}{\partial \theta}$$

is the entropy generation, where $\frac{\partial \text{U}}{\partial \theta}$ is the derivative of the open circuit potential with respect to temperature that you can specify in tabular form.

Joule Heating within the Electrodes and Separator

The total heat generation within the electrode and separator is attributed to the contribution of two different sources: flow of current in the solid phase and flow of current in the electrolyte phase. The amount of energy converted into heat from each of the source terms can be scaled using the conversion factors, β_3, β_4:

$$Q_{\text{tot}} = \beta_3 Q_3 + \beta_4 Q_4,$$

where

$$Q_3 = \mathbf{K}_s^{\text{eff}} \nabla \Phi_s . \nabla \Phi_s$$

is the ohmic loss in the electrode; and

$$Q_4 = \mathbf{K}_e^{\text{eff}} \nabla \Phi_e . \nabla \Phi_e$$

is the ohmic loss in the separator or electrolyte.

Fully Coupled Solution Scheme

The coupled thermal-electrochemical analysis in Abaqus uses an exact implementation of Newton's method, leading to an unsymmetric Jacobian matrix in the form:

$$\begin{bmatrix} K_{\varphi_s\varphi_s} & K_{\varphi_s\theta} & K_{\varphi_s\varphi_e} & K_{\varphi_s C_e} & K_{\varphi_s C_s} \\ K_{\theta\varphi_s} & K_{\theta\theta} & K_{\theta\varphi_e} & K_{\theta C_e} & K_{\theta C_s} \\ K_{\varphi_e\varphi_s} & K_{\varphi_e\theta} & K_{\varphi_e\varphi_e} & K_{\varphi_e C_e} & K_{\varphi_e C_s} \\ K_{C_e\varphi_s} & K_{C_e\theta} & K_{C_e\varphi_e} & K_{C_e C_e} & K_{C_e C_s} \\ K_{C_s\varphi_s} & K_{C_s\theta} & K_{C_s\varphi_e} & K_{C_s C_e} & K_{C_s C_s} \end{bmatrix} \begin{Bmatrix} \Delta\varphi_s \\ \Delta\theta \\ \Delta\varphi_e \\ \Delta C_e \\ \Delta \text{C}_s \end{Bmatrix} = \begin{Bmatrix} R_{\varphi_s} \\ R_\theta \\ R_{\varphi_e} \\ R_{C_e} \\ R_{C_s} \end{Bmatrix}.$$

The coupled thermal-electrochemical-structural analysis in Abaqus also uses an exact implementation of Newton's method, leading to an unsymmetric Jacobian matrix in the form:

$$\begin{bmatrix} K_{uu} & K_{u\varphi_s} & K_{u\theta} & K_{u\varphi_e} & K_{uC_e} & K_{uC_s} \\ K_{\varphi_s u} & K_{\varphi_s\varphi_s} & K_{\varphi_s\theta} & K_{\varphi_s\varphi_e} & K_{\varphi_s C_e} & K_{\varphi_s C_s} \\ K_{\theta u} & K_{\theta\varphi_s} & K_{\theta\theta} & K_{\theta\varphi_e} & K_{\theta C_e} & K_{\theta C_s} \\ K_{\varphi_e u} & K_{\varphi_e\varphi_s} & K_{\varphi_e\theta} & K_{\varphi_e\varphi_e} & K_{\varphi_e C_e} & K_{\varphi_e C_s} \\ K_{C_e u} & K_{C_e\varphi_s} & K_{C_e\theta} & K_{C_e\varphi_e} & K_{C_e C_e} & K_{C_e C_s} \\ K_{C_s u} & K_{C_s\varphi_s} & K_{C_s\theta} & K_{C_s\varphi_e} & K_{C_s C_e} & K_{C_s C_s} \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta\varphi_s \\ \Delta\theta \\ \Delta\varphi_e \\ \Delta \text{C}_e \\ \Delta \text{C}_s \end{Bmatrix} = \begin{Bmatrix} R_u \\ R_{\varphi_s} \\ R_\theta \\ R_{\varphi_e} \\ R_{C_e} \\ R_{C_s} \end{Bmatrix}.$$

Steady-State Analysis

Steady-state analysis provides the steady-state solution by neglecting the transient terms in the continuum scale equations. It can be used to achieve a balanced initial state or to assess conditions in the cell after a long storage period.

In the thermal equation, the internal energy term in the governing heat transfer equation is omitted. Similarly, the transient term is omitted in the diffusion equations for the lithium ion concentration in the solid electrolyte and lithium in the cathode. Electrical transient effects are not included in the equations because they are very rapid compared to the characteristic times of thermal and diffusion effects.

Input File Usage: **COUPLED THERMAL-ELECTROCHEMICAL*, STEADY STATE

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL, STEADY STATE

Transient Analysis

In a transient analysis, the transient effects in the heat transfer and diffusion equations are included in the solution. Electrical transient effects are always omitted because they are very rapid compared to the characteristic times of thermal and mass diffusion effects.

Input File Usage: **COUPLED THERMAL-ELECTROCHEMICAL*

**COUPLED TEMPERATURE-DISPLACEMENT*, ELECTROCHEMICAL

Spurious Oscillations due to Small Time Increments

By default, Abaqus/Standard uses nodal integration of the heat capacity term for first-order elements in a transient analysis, resulting in a lumped treatment for this term. This treatment eliminates nonphysical oscillations; however, it can lead to locally inaccurate solutions, especially in terms of the heat flux for small time increments. If smaller time increments are required, you should use a finer mesh in regions where the temperature changes occur. For more information about time integration involving the temperature degree of freedom, see *Spurious Oscillations due to Small Time Increments* in *Uncoupled Heat Transfer Analysis*.

Controlling the Analysis Based on Solution Variables

A typical battery analysis consists of four or more steps, where the loading is in the following order:

- A charging step, in which a constant current is applied and the voltage is monitored during the step. The step is terminated when the specified cut-off voltage is reached.
- A constant voltage step, in which the voltage is prescribed and the reaction current is monitored. The step is terminated when the current drawn to satify the prescibed voltage condition drops below a specifed threshold value.
- A resting step, in which no current or voltage loads are applied.
- A discharge step, in which a constant current is applied to drain the battery and the voltage is monitored during the step. The step is terminated when the specified cut-off voltage is reached.

You can use step controls (see *Controlling the Analysis Based on the Simulation State in Abaqus/Standard*) in combination with sensors (see *Defining Sensors* to model the workflow described above (by terminating steps based on monitored voltages or currents in specific locations within the model).

Initial Conditions

By default, the initial values of electric potential in the solid, temperature, electric potential in the electrolyte, ion concentration, and species concentration of all nodes are set to zero. You can specify nonzero initial values for the primary solution variables (see *Initial Conditions*).

The typical set of initial conditions includes the initial lithium concentration in the cathode, initial ion concentration in the solid electrolyte, and solid and electrolyte electric potentials in the electrodes. The ion concentration in the electrolyte is typically assumed to be uniform in the cell.

Boundary Conditions

You can prescribe the following boundary conditions:

- Electric potential in the solid, $\varphi_s = \varphi_s(\mathbf{x}, t)$ (degree of freedom 9).
- Electric potential in the electrolyte, $\varphi_e = \varphi_e(\mathbf{x}, t)$ (degree of freedom 32).
- Temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11).
- Ion concentration in the electrolyte, $C_e = C_e(\mathbf{x}, t)$ (degree of freedom 33) at the nodes.
- Species concentration in the electrode, $C_s = C_s(\mathbf{x}, t)$ (degree of freedom 34) at the nodes.

You can specify boundary conditions as functions of time by referring to amplitude curves.

A boundary without any prescribed boundary conditions corresponds to an insulated (zero flux) surface.

The typical boundary condition consists only of grounding (setting to zero) the solid electric potential at the anode. Thermal boundary conditions vary.

Loads

You can apply thermal, electrical, and electrochemical loads in a coupled thermal-electrochemical analysis.

You can prescribe the following types of thermal loads (as described in *Thermal Loads*):

- Concentrated heat flux.
- Body flux and distributed surface flux.
- Convective film and radiation conditions.

You can prescribe the following types of electrical loads on the solid (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of electrical loads on the electrolyte (as described in *Electromagnetic Loads*):

- Concentrated current.
- Distributed surface current densities and body current densities.

You can prescribe the following types of ion concentration loads (as described in *Thermal Loads*):

- Concentrated flux.
- Distributed body flux.

The typical loads include specification of a solid electric flux (current) at the cathode. Thermal boundary conditions vary but typically include convective film on the exterior surfaces. Customarily, no loads are applied on the ion or species concentrations and on the electrolyte potential.

Predefined Fields

Predefined temperature fields are not allowed in coupled thermal-electrochemical analyses. You can use boundary conditions to specify temperatures. You can specify other predefined field variables in a coupled thermal-electrochemical analysis. These values affect only field variable–dependent material properties.

Material Options

The thermal and electrical properties for both the solid and the electrolyte are active in a coupled thermal-electrochemical analysis. All mechanical behavior material models (such as elasticity and plasticity) are ignored in a coupled thermal-electrochemical analysis. The solid electrolyte framework requires that the material definition contain the complete specification of properties required for the solid electrolyte theory, as described below and in the sections that follow. In addition, the material name must begin with "ABQ_EChemSET_" to enable the fully coupled solution between the various fields at the different electrodes. Special-purpose parameter and property tables of type names starting with “ABQ_EChemSET_” are required in these material definitions (see *Parameter Table Type Reference* and *Property Table Type Reference*). In a fully coupled thermal-electrochemical-structural analysis, the material definition must include thermal, electrical, electrochemical, and mechanical properties.

Input File Usage: **MATERIAL*, NAME=ABQ_EChemSET_*name*
**SOLID ELECTROLYTE THEORY*

Defining Properties to Model Solid-State Batteries

You must specify the constants required to define the Butler-Volmer kinetics using different table collections for the anode-separator interface and the cathode-separator interface.These table collections can be used in the surface interaction behavior or in a surface-based load. For the cathode and separator, you must define material properties as defined below.

Interaction at the Anode-Solid Electrolyte Interface

You must define the parameters for the Butler-Volmer kinetics and the open circuit potential for the anode in tabular form as a function of temperature. Optionally, you can specify an Arrhenius form of temperature dependency for I_0^{a-s} (see *Arrhenius Temperature Dependency*.) If Joule heating effects are important, you must also specify the Joule heating losses and the entropy table that is used to define the derivative of the open circuit potential with respect to temperature.

Input File Usage: Use a table collection definition to include all properties (listed below) required to define the Butler-Volmer kinetics:

**TABLE COLLECTION*

Use the following options to define the coefficients of the Butler-Volmer kinetics:

**PARAMETER TABLE*, TYPE=*"ABQ_EChem_LMB_ButlerVolmerInterf"*

k_c, , α_c, α_a, C_e^{ref}, , , S_{fac}, z

Use the following options to define the coefficients of the Arrhenius model:

*PARAMETER TABLE, TYPE="ABQ_EChem_LMB_Arrhenius"

E_a, θ_{ref}

Use the following options to define the open circuit potential for a layer in tabular format:

*PROPERTY TABLE, TYPE="ABQ_EChem_LMB_OCPTabular"

E_{ref}, θ

Use the following options to define the Joule heat fraction at the interface:

*PARAMETER TABLE, TYPE="ABQ_EChem_LMB_Interface_PowerLoss"

β_1, β_2

*PROPERTY TABLE, TYPE="ABQ_EChem_LMB_dUdTEntropy_Tabular"

$\frac{\partial U}{\partial \theta}$, θ

Separator Material Definition

You must define the solid electrolyte properties for the separator region using the tables specified below.

Input File Usage:

Use the following options to define the electrode name, region type (separator), and other electrode properties:

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemSET_Electrode_Definition",
LABEL="ABQ_EChemSET_Electrode_Definition"
"Electrode name", "Region Type"

Use the following options to define the electrolyte name and charge number:

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemSET_Electrolyte",
LABEL="ABQ_EChemSET_Electrolyte"
"Electrolyte name", z

Use the following options to define the electric conductivity of the electrolyte:

*MATERIAL
*PROPERTY TABLE, TYPE="ABQ_EChemSET_Electrolyte_ElecCond_Tabular",

LABEL="ABQ_EChemSET_<*Electrolyte name*>_ElecCond_Tabular"

K_e, C_e,θ, **FV**
*PARAMETER TABLE, TYPE="ABQ_EChemSET_Arrhenius",
LABEL="ABQ_EChemSET_<*Electrolyte name*>_CondArrhenius"

E_a, θ_{ref}

Use the following options to define the diffusion coefficient of the electrolyte:

*MATERIAL
*PROPERTY TABLE, TYPE="ABQ_EChemSET_Electrolyte_DiffTabular",
LABEL="ABQ_EChemSET_<*Electrolyte name*>_DiffTabular"

D_e, C_e, θ, **FV**
*PARAMETER TABLE, TYPE="ABQ_EChemSET_Arrhenius",
LABEL="ABQ_EChemSET_<*Electrolyte name*>_DiffArrhenius"

E_a, θ_{ref}

Use the following option to define the solid transfer reaction:

*MATERIAL
*PARAMETER TABLE,
TYPE="ABQ_EChemSET_Electrolyte_Solid_TransferReaction",
LABEL="ABQ_EChemSET_<*Electrolyte name*>_Solid_TransferReaction"

δ, K_r,C_0

Use the following option to define the Joule heat fraction within the separator region:

*PARAMETER TABLE, TYPE="ABQ_EChemSET_Electrode_PowerLoss"

β_3, β_4

Interaction at the Solid Electrolyte-Cathode Interface

You must define I_0^{s-c} either in terms of parameters or in in tabular form as a function of C_e and C_s. In addition, you must define the open circuit potential for the cathode in tabular form as a function of C_s. Optionally, you can specify an Arrhenius form of the temperature dependency for I_0^{s-c}(see *Arrhenius Temperature Dependency*.) If Joule heating effects are important, you must also specify the Joule heating losses and the entropy table that is used to define the derivative of the open circuit potential with respect to temperature.

Input File Usage: Use a table collection definition to include all properties (listed below) required to define the Butler-Volmer kinetics:

*TABLE COLLECTION

Use the following options to define the coefficients of the Butler-Volmer kinetics:

*PARAMETER TABLE, TYPE="ABQ_EChemSET_ButlerVolmerInterf"

k_c, , α_c, α_a, $C_s^{\min}$, $C_s^{\max}$, C_e^{ref}, S_{fac}, z

Use the following options to specify a tabular form of I_0^{s-c} as a function of C_e and C_s:

*PROPERTY TABLE, TYPE="ABQ_EChemSET_CurrXchgDens_Tabular"

I_0^{s-c}, C_e, C_s

Use the following options to define the coefficients of the Arrhenius model:

*PARAMETER TABLE, TYPE="ABQ_EChemSET_BVArrhenius"

E_a, θ_{ref}

Use the following options to define the open circuit potential for a layer in tabular format:

*PROPERTY TABLE, TYPE="ABQ_EChemSET_OCPTabular"

E_{ref}, C_s

Use the following options to define the Joule heat fraction at the interface:

*PARAMETER TABLE, TYPE="ABQ_EChemSET_Interface_PowerLoss"

β_1, β_2

*PROPERTY TABLE, TYPE="ABQ_EChemSET_dUdTEntropy_Tabular"

$\frac{\partial U}{\partial \theta}$, θ

Cathode Region Material Definition

You must define the properties for the cathode region using the tables specified below.

Input File Usage:

Use the following options to define the electrode name, region type (cathode), and other electrode properties:

*MATERIAL
*PARAMETER TABLE, TYPE="ABQ_EChemSET_Electrode_Definition",
LABEL="ABQ_EChemSET_Electrode_Definition"
"Electrode name", "Region Type"

Use the following options to define the electric conductivity of the solid phase:

*MATERIAL
*PROPERTY TABLE, TYPE="ABQ_EChemSET_Electrode_ElecCond_Tabular",
LABEL="ABQ_EChemSET_<*Electrode name*>_ElecCond_Tabular"

K_s, C_s, θ, **FV**

Use the following options to define the species diffusivity of the solid phase:

*MATERIAL
*PROPERTY TABLE, TYPE="ABQ_EChemSET_Electrode_DiffTabular",
LABEL="ABQ_EChemSET_<*Electrode name*>_DiffTabular"

D_s, C_s, θ, **FV**

Use the following option to define the Joule heat fraction within the cathode region:

*PARAMETER TABLE, TYPE="ABQ_EChemSET_Electrode_PowerLoss"

β_3, β_4

Thermal Material Properties

You must define thermal conductivity for the heat transfer portion of the analysis. In addition, you must define the specific heat for transient problems. Thermal expansion coefficients are not meaningful in a coupled thermal-electrochemical analysis because the deformation of the structure is not considered. You can specify internal heat generation.

Input File Usage: **MATERIAL*
**CONDUCTIVITY*
**DENSITY*
**SPECIFIC HEAT*

Defining a Butler-Volmer Surface Interaction

Abaqus/Standard offers two options to define the Butler-Volmer surface interaction.

When the different regions (that is, the solid lithium anode, solid electrolyte, and solid cathode) are modeled using matching meshes with shared nodes, you can define the Butler-Volmer interaction as a surface-based load on the facets of the anode and cathode elements, respectively.

If the meshes on the two sides of the interface between the solid anode, separator, and solid cathode are dissimilar and do not share nodes, you can use a surface interaction property to define an interface reaction behavior.

For both definitions you must specify the name of the table collection that includes the corresponding surface interaction properties, as described in the previous section.

Input File Usage: Use the following option to define a surface-based Butler-Volmer load:

**STEP*,
**MULTIPHYSICS LOAD*, DEFINITION=BUTLER-VOLMER

Use the following options to define an interface reaction behavior:

**SURFACE INTERACTION*
**INTERFACE REACTION*, TYPE=BUTLER-VOLMER
**CONTACT*
**CONTACT PROPERTY ASSIGNMENT*

Elements

The simultaneous solution in a coupled thermal-electrochemical analysis requires the use of elements that have electric potential in the solid (degree of freedom 9), temperature (degree of freedom 11), electric potential in the electrolyte (degree of freedom 32), ion concentration in the electrolyte (degree of freedom 33), and species concentration in the solid cathode (degree of freedom 34) as nodal variables. Coupled thermal-electrochemical elements are available in Abaqus/Standard only in three dimensions (see *Coupled Thermal-Electrochemical Elements*).

Output

In addition to the output quantities available for the coupled thermal-electric procedure, you can request the following output variables in a coupled thermal-electrochemical analysis.

Nodal output variables:

EPOT	Electric potential in the solid phase.

EPOTE	Electric potential in the solid electrolyte.
NNCE	Ion concentration in the solid electrolyte.
RECURE	Reaction current in the solid electrolyte.
RFLCE	Reaction ion concentration in the solid electrolyte.
NNCS	Species concentration.
RFLCS	Reaction species concentration.

Element Integration point output variables:

CONCS	Species concentration
MFLS	Species concentration flux.

In addition, in a fully coupled thermal-electrochemical-structural analysis you can request the Abaqus/Standard output variables for mechanical degrees of freedom.

References

- Danilov, D., A. H. Niessen, and P. H. L. Notten, "Modeling All-Solid-State Li-Ion Batteries," *Journal of the Electrochemical Society*, vol. 158, no. 3, 2011.
- Tian, H-K., , and Y. Qi, "Simulation of the Effect of Contact Area Loss in All-Solid-State Li-Ion Batteries," *Journal of the Electrochemical Society*, vol. 164, no. 11, 2017.
- Raijmakers, L. H. J., D. L. Danilov, R-A. Eichel, and P. H. L. Notten, "An Advanced All-Solid-State Li-Ion Battery Model," *Electrochimica Acta*, vol. 330, no. 3, 2020.

Parameter Table Type Reference

These parameter table types are used in modeling the electrochemical process in a battery.

In this section:

- *ABQ_EChem_LMB_Arrhenius*
- *ABQ_EChem_LMB_ButlerVolmerInterf*
- *ABQ_EChem_LMB_Interface_PowerLoss*
- *ABQ_EChemPET_Arrhenius*
- *ABQ_EChemPET_CubicSplineC2*
- *ABQ_EChemPET_Electrode_Definition*
- *ABQ_EChemPET_Electrode_FluidVF_BrugExp*
- *ABQ_EChemPET_Electrode_Particle_a_sDistribution*
- *ABQ_EChemPET_Electrode_Particle_ButlerVolmer*
- *ABQ_EChemPET_Electrode_Particle_ElectrodeDissolution*
- *ABQ_EChemPET_Electrode_Particle_IBVUser*
- *ABQ_EChemPET_Electrode_Particle_IDissUser*
- *ABQ_EChemPET_Electrode_Particle_ILPLUser*
- *ABQ_EChemPET_Electrode_Particle_ISEIReformUser*
- *ABQ_EChemPET_Electrode_Particle_ISEIUser*
- *ABQ_EChemPET_Electrode_Particle_Layer_Diffusion*
- *ABQ_EChemPET_Electrode_Particle_Layer_Discretization*
- *ABQ_EChemPET_Electrode_Particle_Layer_OCPUser*
- *ABQ_EChemPET_Electrode_Particle_Layers*
- *ABQ_EChemPET_Electrode_Particle_LPL*
- *ABQ_EChemPET_Electrode_Particle_PowerLoss*
- *ABQ_EChemPET_Electrode_Particle_radiusDistribution*
- *ABQ_EChemPET_Electrode_Particles*
- *ABQ_EChemPET_Electrode_Particle_SEI*
- *ABQ_EChemPET_Electrode_Particle_TR_SEI*
- *ABQ_EChemPET_Electrode_Particle_TR_SOC*
- *ABQ_EChemPET_Electrode_Particle_TR_SpeciesConc*
- *ABQ_EChemPET_Electrode_PowerLoss*
- *ABQ_EChemPET_Electrode_SolidVF_BrugExp*
- *ABQ_EChemPET_Electrode_Swelling*
- *ABQ_EChemPET_Electrolyte*
- *ABQ_EChemPET_Electrolyte_TR*
- *ABQ_EChemSET_ButlerVolmerInterf*
- *ABQ_EChemSET_BVArrhenius*
- *ABQ_EChemSET_Electrode_Definition*
- *ABQ_EChemSET_Electrode_PowerLoss*
- *ABQ_EChemSET_Electrolyte*
- *ABQ_EChemSET_Electrolyte_Solid_TransferReaction*
- *ABQ_EChemSET_Interface_PowerLoss*

ABQ_EChem_LMB_Arrhenius

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This parameter table type is used in modeling the solid electrodes in lithium metal batteries to specify the coefficients to use in the Arrhenius equation for modeling temperature difference.

Optionally, you can include a parameter table of type `"ABQ_EChem_LMB_Arrhenius"` in the table collection definition to specify the coefficients to use in the Arrhenius equation for modeling temperature difference. You can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChem_LMB_Arrhenius", PARAMETERS=2
 FLOAT, 0.0, "E_a", , ,
 FLOAT, 0.0, "θ_ref", , ,
```

Parameters

E_a

Activation energy to use in the Arrhenius equation, $e^{\frac{E_a}{R}\left(\frac{1}{(\theta-\theta^z)}-\frac{1}{(\theta_{\text{ref}}-\theta^z)}\right)}$, where θ^z is the absolute zero temperature.

Default value is zero.

θ_{ref}

Reference temperature.

Default value is zero.

ABQ_EChem_LMB_ButlerVolmerInterf

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This parameter table type is used in modeling the solid electrodes in lithium metal batteries to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

You must include a parameter table of type `"ABQ_EChem_LMB_ButlerVolmerInterf"` in the table collection definition to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChem_LMB_ButlerVolmerInterf", PARAMETERS=9

 FLOAT, , "k_c", , ,
 FLOAT, , "k_a", , ,
 FLOAT, , "α_c", , ,
 FLOAT, , "α_a", , ,
 FLOAT, , "C_e^ref", , ,
 FLOAT, , "R_SEI", , ,
 FLOAT, , "Stress_coef", , ,
 FLOAT, , "S_fac", , ,
 FLOAT, , "z", , ,
```

Parameters

k_c	Cathodic rate constant.
k_a	Anodic rate constant.
α_c	Cathodic transfer coefficient.
α_a	Anodic transfer coefficient.
C_e^{ref}	Reference value of the lithium concentration in the electrolyte.
R_{SEI}	Resistance value of the solid electrolyte interphase (SEI) layer.
$\text{Stress}_{\text{coef}}$	Partial molar volume coefficient for the hydrostatic stress at the particle surface.
S_{fac}	Surface roughness coefficient that captures the increase in surface area compared to geometric surface area.
z	Charge number of electrode.

ABQ_EChem_LMB_Interface_PowerLoss

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a solid electrode battery to specify the Joule heat fraction losses at the solid electrode-electrolyte interface and due to entropy generation.

You must include a parameter table of type `"ABQ_EChem_LMB_Interface_PowerLoss"` in the table collection definition to specify the Joule heat fraction losses at the solid electrode-electrolyte interface and due to entropy generation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChem_LMB_Interface_PowerLoss", PARAMETERS=2

FLOAT, 0.0, "β1", , ,
FLOAT, 0.0, "β2", , ,
```

Parameters

β_1
: Fraction of ohmic loss at the solid-liquid interface.

  Default value is zero.

β_2
: Fraction of entropy generation.

  Default value is zero.

ABQ_EChemPET_Arrhenius

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the coefficients to use in the Arrhenius equation for modeling temperature difference.

Optionally, you can include a parameter table of type `"ABQ_EChemPET_Arrhenius"` in the material definition to specify the coefficients to use in the Arrhenius equation for modeling temperature difference. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Arrhenius", PARAMETERS=2
 FLOAT, 0.0, "E_a", , ,
 FLOAT, 0.0, "θ_ref", , ,
```

Parameters

E_a

Activation energy to use in the Arrhenius equation, $e^{\frac{E_a}{R}\left(\frac{1}{(\theta-\theta^z)}-\frac{1}{(\theta_{\text{ref}}-\theta^z)}\right)}$, where θ^z is the absolute zero temperature.

Default value is zero.

θ_{ref}

Reference temperature.

Default value is zero.

ABQ_EChemPET_CubicSplineC2

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the material curve as a cubic spline over equal interval sizes.

You must include a parameter table of type `"ABQ_EChemPET_CubicSplineC2"` in the material definition to specify the material curve as a cubic spline over equal interval sizes. Each interval is fitted with a cubic spline equation of the form $\mathbf{y(x)} = C_3(x - x_i)^3 + C_2(x - x_i)^2 + C_1(x - x_i) + C_0$.

Repeat the parameter table data lines as needed to specify the coefficients of spline interpolation for all intervals covering the full range of the material curve.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_CubicSplineC2", PARAMETERS=2
 FLOAT, , "Xi", , ,
 FLOAT, , "Xi+1", , ,
 FLOAT, , "C3", , ,
 FLOAT, , "C2", , ,
 FLOAT, , "C1", , ,
 FLOAT, , "C0", , ,
```

Parameters

X_i	Start value of the interval over which the specified coefficients are valid for the cubic spline.
X_{i+1}	End value of the interval over which the specified coefficients are valid for the cubic spline.
C_3	Coefficient used in the cubic spline interpolation.
C_2	Coefficient used in the cubic spline interpolation.
C_1	Coefficient used in the cubic spline interpolation.
C_0	Coefficient used in the cubic spline interpolation.

ABQ_EChemPET_Electrode_Definition

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the material data for the electrode region.

You must include a parameter table of type "ABQ_EChemPET_Electrode_Definition" in the material definition to specify the material data for the electrode region. The region name must be unique and is used in the labels of subsequent electrode material property definitions.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Definition",
PARAMETERS=12
 STRING, "  ", "Region Name", , ,
 STRING, "  ", "Region Type", , "Anode|Cathode|Separator",
 FLOAT, 0.0, "ε_s", , ,
 FLOAT, 0.0, "ε_e", , ,
 FLOAT, 0.0, "ε_b", , ,
 FLOAT, 0.0, "ε_inactive", , ,
 FLOAT, 0.0, "Utilization Fraction", , ,
 FLOAT, 0.0, "unused", , ,
 FLOAT, 0.0, "α_z", , ,
 FLOAT, 0.0, "α_x", , ,
 FLOAT, 0.0, "α_y", , ,
 FLOAT, 0.0, "Conv_coef", , ,
```

Parameters

Region Name	Unique name to identify the electrode region.
Region Type	Set to "Anode" to specify that the material definitions are for the anode region. Set to "Cathode" to specify that the material definitions are for the cathode region. Set to "Separator" to specify that the material definitions are for the separator region.
ε_s	Volume fraction of the solid phase in the electrode.
ε_e	Volume fraction of the liquid phase or electrolyte in the electrode.
ε_b	Volume fraction of the binder material in the electrode.
$\varepsilon_{\text{inactive}}$	Volume fraction of the inactive solid phase material in the electrode.

Utilization Fraction	Fraction of utilization of the cathode and anode regions.
α_z	Bruggeman exponent in the thickness direction.
α_x	Bruggeman exponent in the manufacturing direction.
α_y	Bruggeman exponent in the transverse direction.
$\mathrm{Conv}_{\mathrm{coef}}$	Set this value equal to 1.0 when the particle swelling and convection effects of the electrolyte due to swelling should be modeled.

ABQ_EChemPET_Electrode_FluidVF_BrugExp

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the Bruggeman exponents to be used for computing the effective ionic diffusivity and ionic conductivity of the fluid phase.

Optionally, you can include a parameter table of type `"ABQ_EChemPET_Electrode_FluidVF_BrugExp"` in the material definition to specify the Bruggeman exponents to be used for computing the effective ionic diffusivity and ionic conductivity of the electrolyte phase. If this parameter table is defined, the Bruggeman exponent values specified under *ABQ_EChemPET_Electrode_Definition* will not be used for the effective ionic diffusivity and ionic conductivity of the fluid phase. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_SolidVF_BrugExp",
PARAMETERS=3
 FLOAT, 0.0, "α_z^e", , ,
 FLOAT, 0.0, "α_x^e", , ,
 FLOAT, 0.0, "α_y^e", , ,
```

Parameters

α_z^e Bruggeman exponent in the thickness direction for the electrolyte phase.

α_x^e Bruggeman exponent in the manufacturing direction for the electrolyte phase.

α_y^e Bruggeman exponent in the transverse direction for the electrolyte phase.

ABQ_EChemPET_Electrode_Particle_a_sDistribution

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used to include a distribution of particle wetted surface area per unit volume using a predefined distribution table.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrode_Particle_a_sDistribution` in the material definition to specify a distribution for a_s.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_a_sDistribution", PARAMETERS=1
 STRING, , "Distribution table name", , ,
```

Parameters

Distribution Table Name	Set to the name of a previously defined distribution table to specify a_s.

ABQ_EChemPET_Electrode_Particle_ButlerVolmer

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

You must include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_ButlerVolmer"` in the material definition to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_ButlerVolmer",
 PARAMETERS=8
  FLOAT, , "k_c", , ,
  FLOAT, , "k_a", , ,
  FLOAT, , "α_c", , ,
  FLOAT, , "α_a", , ,
  FLOAT, , "C_e^ref", , ,
  FLOAT, , "R_SEI", , ,
  FLOAT, , "Flag_stressCoef^compute", , ,
  FLOAT, , "a_s", , ,
```

Parameters

k_c	Cathodic rate constant.
k_a	Anodic rate constant.
α_c	Cathodic transfer coefficient.
α_a	Anodic transfer coefficient.
C_e^{ref}	Reference value of the lithium concentration in the electrolyte.
R_{SEI}	Resistance value of the solid electrolyte interphase (SEI) layer.
$Flag_{stressCoef}^{compute}$	Set this parameter to a nonzero value to include the effects of mechanical stress on the electrical response of an active electrode particle.
a_s	Set this value to `0.0` to use the value computed by Abaqus. If you specify a nonzero value, the specified value is used for the wetted surface area of the particle.

ABQ_EChemPET_Electrode_Particle_ElectrodeDissolution

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling one of the aging mechanisms, namely, electrode dissolution in a battery.

Optionally, you can include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_ElectrodeDissolution"` in the material definition to specify the total length of the cathode. This also triggers the computation of the electrode dissolution aging mechanism in the battery simulation. You can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_ElectrodeDissolution", PARAMETERS=1
 FLOAT, 0.0, "l_cathode", , ,
```

Parameters

$l_{cathode}$	Total thickness of the cathode. Default value is zero.

ABQ_EChemPET_Electrode_Particle_IBVUser

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to activate the definition of a user-defined Butler-Volmer current density at the microscale particle-electrolyte interface using user subroutine UECHEMECDBV.

You can optionally include a parameter table of type "ABQ_EChemPET_Electrode_Particle_IBVUser" in the material definition to specify the user-defined Butler-Volmer current at the microscale particle-electrolyte interface.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_IBVUser",
PARAMETERS=1
 Integer, 0, "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UECHEMECDBV*.

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_IDissUser

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to activate the definition of a user-defined electrode dissolution current density at the microscale particle-electrolyte interface using user subroutine UECHEMECDDISS.

You can optionally include a parameter table of type "ABQ_EChemPET_Electrode_Particle_IDissUser" in the material definition to specify the user-defined dissolution current at the microscale particle-electrolyte interface.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_IDissUser",
PARAMETERS=1
 Integer, , "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UECHEMECDDISS*.

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_ILPLUser

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a battery t0 activate the definition of a user-defined plated lithium (LPL) current density at the microscale particle-electrolyte interface using user subroutine UECHEMECDLPL.

You can optionally include a parameter table of type "ABQ_EChemPET_Electrode_Particle_ILPLUser" in the material definition to specify the user-defined plated lithium current at the microscale particle-electrolyte interface.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_ILPLUser",
PARAMETERS=1
 Integer, , "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UECHEMECDLPL*.

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_ISEIReformUser

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to activate the definition of a user-defined Solid-Electrolyte Interface (SEI) reformation current density at the microscale particle-electrolyte interface using user subroutine UECHEMECDSEIREFORM.

You can optionally include a parameter table of type "ABQ_EChemPET_Electrode_Particle_ISEIReformUser" in the material definition to specify the user-defined SEI reformation current at the microscale particle-electrolyte interface.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_ISEIReformUser",
 PARAMETERS=1
 Integer, , "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UECHEMECDSEIREFORM*

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_ISEIUser

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to activate the definition of a user-defined Solid-Electrolyte Interface (SEI) current density at the microscale particle-electrolyte interface using user subroutine `UECHEMECDSEI`.

You can optionally include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_ISEIUser"` in the material definition to specify the user-defined SEI current at the microscale particle-electrolyte interface.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_ISEIUser",
PARAMETERS=1
 Integer, , "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UECHEMECDSEI*.

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_Layer_Diffusion

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the diffusion characteristics of a particle layer.

You must include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_Layer_Diffusion"` in the material definition to specify the diffusion characteristics of a particle layer.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_Diffusion", PARAMETERS=3

 FLOAT, 0.0 , "C_{s,P}^{max}", , ,
 STRING, " ", "Diffusion Data Type", ,
"DiffusionModel_Tabular|DiffusionModel_LogTabular|DiffusionModel_CoeffSpline",

 STRING, " ", "Chemical Diffusion Type", , "DiffusionModel_ChemoMech",
```

Parameters

$C_{s,P}^{\max}$	Theoretical maximum lithium capacity of the microscale particle.
Diffusion Data Type	Set to `"DiffusionModel_Tabular"` to input diffusivity as a function of the normalized concentration in the particle in tabular form. Set to `"DiffusionModel_LogTabular"` to input $\mathbf{Log_{10}}$ of the diffusivity as a function of the normalized concentration in the particle in tabular form. Set to `"DiffusionModel_CoeffSpline"` to input diffusivity as a function of the normalized concentration in a cubic spline format.
Chemical Diffusion Type	Set to `"DiffusionModel_ChemoMech"`.

ABQ_EChemPET_Electrode_Particle_Layer_Discretization

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the mesh on the microscale particle.

You must include a parameter table of type "ABQ_EChemPET_Electrode_Particle_Layer_Discretization" in the material definition to specify the mesh on the microscale particle.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_Discretization", PARAMETERS=2
 INTEGER, , "N_elemPerLayer", , ,
 STRING, " ", "Mesh Type", , "Uniform|Quadratic|EquiVolumetric",
```

Parameters

$N_{elemPerLayer}$	Number of elements to use to mesh the layer. Set this value between 1 and 100. You must not exceed a total of 100 elements in a single particle.
Mesh Type	Set to "Uniform" for a uniform mesh in the layer.
	Set to "EquiVolumetric" for a mesh where all the elements have equal volume.
	Set to "Quadratic" for a mesh that is biased toward the outer part of the layer.

ABQ_EChemPET_Electrode_Particle_Layer_OCPUser

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to activate the definition of a user-defined open circuit potential for a particle layer using user subroutine `UEREF`. This open circuit potential value is used in the Butler-Volmer current density at the microscale particle-electrolyte interface calculations.

You can optionally include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_Layer_OCPUser"` in the material definition to specify a user-defined OCP value for a particle layer material.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_Layer_OCPUser",
 PARAMETERS=1
 Integer, , "Activate Usersubroutine usage", , ,
```

Parameters

Activate Usersubroutine usage

Set to **1** to activate usage of user subroutine *UEREF*.

Default value is **0**.

ABQ_EChemPET_Electrode_Particle_Layers

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the material layer name and the weight fraction for the layer.

You must include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_Layers"` in the material definition to specify the material layer name and the weight fraction for the layer. The layer name must be unique and is used in the labels of subsequent layer-specific material definitions.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_Layers",
PARAMETERS=2
 STRING, " ", "Layer Name", , ,
 FLOAT, , "Weight Fraction of Layer", , ,
```

Parameters

Layer Name	Unique name to identify the material region of the particle layer.
Weight Fraction of Layer	Set to `1.0` if only one layer is present in the particle. If multiple layers are present, set between `0.0` and `1.0`, depending on the weight ratio of each layer to the total weight of the particle.

ABQ_EChemPET_Electrode_Particle_LPL

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling one of the aging mechanisms, namely, lithium plating in a battery.

Optionally, you can include a parameter table of type "ABQ_EChemPET_Electrode_Particle_LPL" in the material definition to specify some of the parameters used in the lithium plating computation. This also triggers the computation of the lithium plating aging mechanism in the battery simulation. You can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_LPL",
PARAMETERS=4
 FLOAT, 0.0, "α_c,LPL", , ,
 FLOAT, 0.0, "C^0_LPL", , ,
 FLOAT, 0.0, "ρ_LPL", , ,
 FLOAT, 0.0, "M_LPL", , ,
```

Parameters

$\alpha_{c,LPL}$	Cathodic transfer coefficient for lithium plating. Default value is zero.
C^0_{LPL}	Initial plated lithium concentration. Default value is zero.
ρ_{LPL}	Molar density of plated lithium. Default value is zero.
M_{LPL}	Molar weight of plated lithium. Default value is zero.

ABQ_EChemPET_Electrode_Particle_PowerLoss

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the Joule heat fraction losses at the microscale particle-electrolyte interface and due to entropy generation.

You must include a parameter table of type `"ABQ_EChemPET_Electrode_Particle_PowerLoss"` in the material definition to specify the Joule heat fraction losses at the microscale particle-electrolyte interface and due to entropy generation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_PowerLoss",
PARAMETERS=2
 FLOAT, 0.0, "β1", , ,
 FLOAT, 0.0, "β2", , ,
```

Parameters

β_1
: Fraction of ohmic loss at the solid-liquid interface.

  Default value is zero.

β_2
: Fraction of entropy generation.

  Default value is zero.

ABQ_EChemPET_Electrode_Particle_radiusDistribution

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used to include a distribution of particle radius using a predefined distribution table.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrode_Particle_radiusDistribution` in the material definition to specify a distribution for particle radius.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_radiusDistribution", PARAMETERS=1
 STRING, , "Distribution table name", , ,
```

Parameters

Distribution Table Name	Set to the name of a previously defined distribution table to define the radius of the particle.

ABQ_EChemPET_Electrode_Particles

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify a unique particle name and additional particle information.

You must include a parameter table of type "ABQ_EChemPET_Electrode_Particles" in the material definition to specify a particle name and additional particle information. The particle name must be unique and is used in the labels of subsequent particle-specific material definitions. If you have multiple particles in a region, you can repeat the parameter table data lines.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particles", PARAMETERS=5

STRING, " " , "Particle Name", ,  ,
FLOAT, , "R_P", , ,
FLOAT, , "ε_sParticle", , ,
FLOAT, , "C_s,P,_Init", , ,
STRING, "  " , "Formulation Type", , "Formulation_Concentration",
```

Parameters

Particle Name	Unique name to identify the microscale particle.
R_P	Outer radius of the solid microscale particle.
$\varepsilon_{\text{sParticle}}$	Volume fraction of the microscale particle with respect to the solid volume fraction of the electrode. Set to `1.0` if only one particle is present.
$C_{s,P,\text{_Init}}$	Initial concentration of lithium in the microscale particle.
Formulation Type	Set to "Formulation_Concentration".

ABQ_EChemPET_Electrode_Particle_SEI

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This parameter table type is used in modeling one of the aging mechanisms, namely, Solid Electrolyte Interface in a battery.

Optionally, you can include a parameter table of type "ABQ_EChemPET_Electrode_Particle_SEI" in the material definition to specify some of the parameters used in the SEI aging computation. This also triggers the computation of the SEI aging mechanism in the battery simulation. You can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_SEI",
PARAMETERS=10
 FLOAT, 0.0, "α_{a,SEI}", , ,
 FLOAT, 0.0, "α_{c,SEI}", , ,
 FLOAT, 0.0, "K_{0,SEI}", , ,
 FLOAT, 0.0,
"C_{EC}^{bulk}                                                                ",
 , ,
 FLOAT, 0.0, "C_{EC,0}^{s}", , ,
 FLOAT, 0.0,
"C_{0,SEI}                                                                    ",
 , ,
 FLOAT, 0.0, "β", , ,
 FLOAT, 0.0, "ρ_{SEI}", , ,
 FLOAT, 0.0, "M_{SEI}", , ,
 FLOAT, 0.0, "z", , ,
```

Parameters

$\alpha_{a,SEI}$	Anodic transfer coefficient for the SEI aging computation. Default value is zero.
$\alpha_{c,SEI}$	Cathodic transfer coefficient for the SEI aging computation. Default value is zero.
$K_{0,SEI}$	Kinetic rate constant of SEI. Default value is zero.

C_{EC}^{bulk} Bulk concentration of ethylene carbonate in the electrolyte.
Default value is zero.

$C_{EC,0}^{s}$ Initial surface concentration of ethylene carbonate on the surface of the particle.
Default value is zero.

$C_{0,SEI}$ Initial SEI concentration.
Default value is zero.

β Fraction of plated lithium that can convert to SEI.
Default value is zero.

ρ_{SEI}. Molar density of SEI.
Default value is zero.

M_{SEI}. Molar weight of SEI.
Default value is zero.

z. Charge number of SEI.
Default value is zero.

ABQ_EChemPET_Electrode_Particle_TR_SEI

Products: Abaqus/Standard

References:

- *Modeling Thermal Runaway in Batteries*

Overview

This parameter table type is used to model the heat generated during a thermal runaway process due to decomposition of the solid electrolyte interface (SEI) layer.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrode_Particle_TR_SOC` in the material definition to include the effects of SEI decomposition during thermal runaway.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_TR_SEI",
PARAMETERS=6
 FLOAT, , "Frequency factor for SEI decomposition", , ,
 FLOAT, , "Activation energy for SEI decomposition", , ,
 FLOAT, , "Enthalpy of SEI decomposition", , ,
 FLOAT, , "Density of SEI", , ,
 FLOAT, , "Initial SEI at start of analysis", , ,
 FLOAT, , "Fraction of heat energy generated from SEI decomposition", , ,
```

Parameters

A_{sei}	Frequency factor for SEI decomposition in units of $\frac{1}{T}$.
E_{sei}	Activation energy for SEI decomposition.
h_{sei}	Enthalpy of SEI decomposition.
ρ_{sei}	Density of SEI.
SEI_{init}	Initial concentration of SEI at the start of the analysis.
η_{sei}	Fraction of heat energy generated from SEI decomposition.

ABQ_EChemPET_Electrode_Particle_TR_SOC

Products: Abaqus/Standard

References:

- *Modeling Thermal Runaway in Batteries*

Overview

This parameter table type is used in modeling the heat generated during a thermal runaway process due to decomposition of the state of charge.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrode_Particle_TR_SOC` in the material definition to include the effects of state of charge (SOC) decomposition during thermal runaway.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_TR_SOC",
PARAMETERS=5
 FLOAT, , "Frequency factor for SOC decomposition", , ,
 FLOAT, , "Activation energy for SOC decomposition", , ,
 FLOAT, , "Enthalpy of SOC decomposition", , ,
 FLOAT, , "Density of SOC", , ,
 FLOAT, , "Fraction of heat energy generated from SOC decomposition", , ,
```

Parameters

A_{soc}	Frequency factor for SOC decomposition in units of $\frac{1}{T}$.
E_{soc}	Activation energy for SOC decomposition.
h_{soc}	Enthalpy of SOC decomposition.
ρ_{soc}	Density of SOC.
η_{soc}	Fraction of heat energy generated from SOC decomposition.

ABQ_EChemPET_Electrode_Particle_TR_SpeciesConc

Products: Abaqus/Standard

References:

- *Modeling Thermal Runaway in Batteries*

Overview

This parameter table type is used in modeling the heat generated during a thermal runaway process due to decomposition of the intercalated lithium in the particles.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrode_Particle_TR_SpeciesConc` in the material definition to include the effects of decomposition of intercalated lithium in particles during thermal runaway.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_TR_SpeciesConc",
 PARAMETERS=5
 FLOAT, , "Frequency factor for particle species concentration
decomposition", , ,
 FLOAT, , "Activation energy for particle species concentration
decomposition", , ,
 FLOAT, , "Enthalpy of particle species concentration  decomposition", , ,

 FLOAT, , "Density of particle species concentration ", , ,
 FLOAT, , "Dimensionless measure of SEI thickness", , ,
 FLOAT, , "Fraction of heat energy generated from particle species
concentration decomposition", , ,
```

Parameters

A_{spc}	Frequency factor for particle species concentration decomposition in units of $\frac{1}{T}$.
E_{spc}	Activation energy for particle species concentration decomposition.
h_{spc}	Enthalpy of particle species concentration decomposition.
ρ_{spc}	Density of particle species concentration.
z_0	Dimensionless measure of SEI thickness.
η_{spc}	Fraction of heat energy generated from particle species concentration decomposition.

ABQ_EChemPET_Electrode_PowerLoss

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the Joule heat fraction losses at the macroscale level due to the flow of electrical energy in the solid and liquid phases and due to ion diffusion.

You must include a parameter table of type `"ABQ_EChemPET_Electrode_PowerLoss"` in the material definition to specify the Joule heat fraction losses at the macroscale level due to the flow of electrical energy in the solid and liquid phases and due to ion diffusion.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_PowerLoss", PARAMETERS=3

FLOAT, 0.0, "β3", , ,
FLOAT, 0.0, "β4", , ,
FLOAT, 0.0, "β5", , ,
```

Parameters

β_3
Fraction of electrical energy flowing through the solid phase that is converted to heat.
Default value is zero.

β_4
Fraction of electrical energy flowing through the electrolyte that is converted to heat.
Default value is zero.

β_5
Fraction of ohmic loss due to the diffusion of ions in the electrolyte.
Default value is zero.

ABQ_EChemPET_Electrode_SolidVF_BrugExp

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the Bruggeman exponents to be used for computing the effective electrical conductivity of the solid phase.

Optionally, you can include a parameter table of type `"ABQ_EChemPET_Electrode_SolidVF_BrugExp"` in the material definition to specify the Bruggeman exponents to be used for computing the effective electrical conductivity of the solid phase. If this parameter table is defined, the Bruggeman exponent values specified under *ABQ_EChemPET_Electrode_Definition* will not be used for the effective electrical conductivity of the solid phase. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_SolidVF_BrugExp",
PARAMETERS=3
 FLOAT, 0.0, "α_z^s", , ,
 FLOAT, 0.0, "α_x^s", , ,
 FLOAT, 0.0, "α_y^s", , ,
```

Parameters

α_z^s Bruggeman exponent in the thickness direction for the solid phase.

α_x^s Bruggeman exponent in the manufacturing direction for the solid phase.

α_y^s Bruggeman exponent in the transverse direction for the solid phase.

ABQ_EChemPET_Electrode_Swelling

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*
- *Fully Coupled Thermal-Electrochemical-Structural Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify anisotropic scaling factors in the X-, Y-, and Z-directions.

You can include a parameter table of type `"ABQ_EChemPET_Electrode_Swelling"` in the material definition to specify a scaling factor for the particle swelling effects in the X-, Y-, and Z-directions.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Swelling", PARAMETERS=3

FLOAT, , "X_scale", , ,
FLOAT, , "Y_scale", , ,
FLOAT, , "Z_scale", , ,
```

Parameters

X_{scale}	Scale factor to scale the computed eigenstrain in the X-direction.
Y_{scale}	Scale factor to scale the computed eigenstrain in the Y-direction.
Z_{scale}	Scale factor to scale the computed eigenstrain in the Z-direction.

ABQ_EChemPET_Electrolyte

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This parameter table type is used in modeling the electrochemical process in a battery to specify the electrolyte region name to use in the label names of subsequent electrolyte property definitions and to define the charge number.

You must include a parameter table of type `"ABQ_EChemPET_Electrolyte"` in the material definition to specify the electrolyte region name to use in the label names of subsequent electrolyte property definitions and to define the charge number.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte", PARAMETERS=3
 STRING, " ", "Electrolyte Name ", , ,
 FLOAT, 0.0, "z", , ,
 FLOAT, 0.0, "unused", , ,
```

Parameters

Electrolyte Name	Unique name to identify the electrolyte.
z	Charge number of the lithium ion battery.

ABQ_EChemPET_Electrolyte_TR

Products: Abaqus/Standard

References:

- *Modeling Thermal Runaway in Batteries*

Overview

This parameter table type is used in modeling the heat generated during a thermal runaway process due to decomposition of the electrolyte.

You can optionally include a parameter table of type `ABQ_EChemPET_Electrolyte_TR` in the material definition to include the effects of electrolyte decomposition during thermal runaway.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte_TR", PARAMETERS=5
 FLOAT, , "Frequency factor for electrolyte decomposition", , ,
 FLOAT, , "Activation energy for electrolyte decomposition", , ,
 FLOAT, , "Enthalpy of electrolyte decomposition", , ,
 FLOAT, , "Density of electrolyte for electrolyte decomposition", , ,
 FLOAT, , "Fraction of heat energy generated from electrolyte decomposition",
 , ,
```

Parameters

A_e	Frequency factor for electrolyte decomposition in units of $\frac{1}{T}$.
E_e	Activation energy for electrolyte decomposition.
h_e	Enthalpy of electrolyte decomposition.
ρ_e	Density of electrolyte for electrolyte decomposition.
η_e	Fraction of heat energy generated from electrolyte decomposition.

ABQ_EChemSET_ButlerVolmerInterf

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid electrolyte interface in solid state batteries to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

You must include a parameter table of type `"ABQ_EChemSET_ButlerVolmerInterf"` in the table collection definition to specify the coefficients to use to compute the current density using the Butler-Volmer equation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_ButlerVolmerInterf", PARAMETERS=10

 FLOAT, , "k_c", , ,
 FLOAT, , "  Unused ", , ,
 FLOAT, , "α_c", , ,
 FLOAT, , "α_a", , ,
 FLOAT, , "C_s^min", , ,
 FLOAT, , "C_s^max", , ,
 FLOAT, , "C_e^ref", , ,
 FLOAT, , "S_fac", , ,
 FLOAT, , "  Unused ", , ,
 FLOAT, , "z", , ,
```

Parameters

k_c	Cathodic rate constant.
α_c	Cathodic transfer coefficient.
α_a	Anodic transfer coefficient.
C_s^{min}	Minimum possible concentration of lithium in electrode.
C_s^{max}	Maximum possible concentration of lithium in electrode.
C_e^{ref}	Reference lithium ion concentration in electrolyte.
S_{fac}	Surface roughness coefficient that captures the increase in surface area compared to geometric surface area.
z	Charge number of electrode.

ABQ_EChemSET_BVArrhenius

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid electrolyte interface in solid state batteries to specify the coefficients to use in the Arrhenius equation for modeling temperature difference.

Optionally, you can include a parameter table of type `"ABQ_EChemSET_BVArrhenius"` in the table collection definition to specify the coefficients to use in the Arrhenius equation for modeling temperature difference. You can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_BVArrhenius", PARAMETERS=2
 FLOAT, 0.0, "E_a", , ,
 FLOAT, 0.0, "θ_ref", , ,
```

Parameters

E_a

Activation energy to use in the Arrhenius equation, $e^{\frac{E_a}{R}\left(\frac{1}{(\theta-\theta^z)}-\frac{1}{(\theta_{\text{ref}}-\theta^z)}\right)}$, where θ^z is the absolute zero temperature.

Default value is zero.

θ_{ref}

Reference temperature.

Default value is zero.

ABQ_EChemSET_Electrode_Definition

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid state batteries with solid electrolytes to specify the material data for the electrode region.

You must include a parameter table of type "ABQ_EChemSET_Electrode_Definition" in the material definition to specify the material data for the electrode region. The region name must be unique and is used in the labels of subsequent electrode material property definitions.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_Electrode_Definition", PARAMETERS=3

STRING, "  ", "Region Name", , ,
STRING, "  ", "Region Type", , "Anode|Cathode|Separator",
FLOAT, 0.0, "Unused", , ,
```

Parameters

Region Name Unique name to identify the electrode region.

Region Type Set to "Anode" to specify that the material definitions are for the anode region.

Set to "Cathode" to specify that the material definitions are for the cathode region.

Set to "Separator" to specify that the material definitions are for the separator region.

ABQ_EChemSET_Electrode_PowerLoss

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid electrolyte interface in a solid state battery to specify the joule heat fraction losses within the electrode and separator regions.

You must include a parameter table of type `"ABQ_EChemSET_Electrode_PowerLoss"` in the table collection definition to specify the Joule heat fraction losses within the electrode and separator regions.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_Electrode_PowerLoss", PARAMETERS=2

FLOAT, 0.0, "β3", , ,
FLOAT, 0.0, "β4", , ,
```

Parameters

β_3
Fraction of ohmic loss in the electrode.
Default value is zero.

β_4
Fraction of ohmic loss in the electrolyte.
Default value is zero.

ABQ_EChemSET_Electrolyte

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the electrochemical process in a solid electrolyte battery to specify the electrolyte region name to use in the label names of subsequent electrolyte property definitions and to define the charge number.

You must include a parameter table of type `"ABQ_EChemSET_Electrolyte"` in the material definition to specify the electrolyte region name to use in the label names of subsequent electrolyte property definitions and to define the charge number.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_Electrolyte", PARAMETERS=3
 STRING, " ", "Electrolyte Name ", , ,
 FLOAT, 0.0, "z", , ,
 FLOAT, 0.0, "unused", , ,
```

Parameters

Electrolyte Name	Unique name to identify the electrolyte.
z	Charge number of the lithium ion battery.

ABQ_EChemSET_Electrolyte_Solid_TransferReaction

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid electrolyte in a battery to specify the transfer reaction parameters.

You must include a parameter table of type `"ABQ_EChemSET_Electrolyte_Solid_TransferReaction"` in the material definition to specify the transfer reaction properties.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_EChemSET_Electrolyte_Solid_TransferReaction", PARAMETERS=3
 FLOAT, 0.0, "δ", , ,
 FLOAT, 0.0, "K_r", , ,
 FLOAT, 0.0, "C_0", , ,
```

Parameters

δ	Fraction of mobile lithium.
K_r	Combination rate (or) reverse reaction rate.
C_0	Total amount of lithium.

ABQ_EChemSET_Interface_PowerLoss

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This parameter table type is used in modeling the solid electrolyte interface in a solid state battery to specify the joule heat fraction losses at the solid electrode-electrolyte interface and due to entropy generation.

You must include a parameter table of type `"ABQ_EChemSET_Interface_PowerLoss"` in the table collection definition to specify the joule heat fraction losses at the solid electrode-solid electrolyte interface and due to entropy generation.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_EChemSET_Interface_PowerLoss", PARAMETERS=2

FLOAT, 0.0, "β1", , ,
FLOAT, 0.0, "β2", , ,
```

Parameters

β_1
: Fraction of ohmic loss at the solid electrode-solid electrolyte interface.

  Default value is zero.

β_2
: Fraction of entropy generation.

  Default value is zero.

Property Table Type Reference

These property table types are used in modeling the electrochemical process in a battery.

In this section:

- *ABQ_EChem_LMB_CurrXchgDens_Tabular*
- *ABQ_EChem_LMB_dUdTEntropy_Tabular*
- *ABQ_EChem_LMB_OCPTabular*
- *ABQ_EChemPET_Electrode_ElecCond_Tabular*
- *ABQ_EChemPET_Electrode_Particle_CurrXchgDens_CrackSEI_Tabular*
- *ABQ_EChemPET_Electrode_Particle_CurrXchgDens_ElecDiss_Tabular*
- *ABQ_EChemPET_Electrode_Particle_CurrXchgDens_LPL_Tabular*
- *ABQ_EChemPET_Electrode_Particle_CurrXchgDens_SEI_Tabular*
- *ABQ_EChemPET_Electrode_Particle_CurrXchgDens_Tabular*
- *ABQ_EChemPET_Electrode_Particle_dUdTEntropy_Tabular*
- *ABQ_EChemPET_Electrode_Particle_ElecDiss_OCPTabular*
- *ABQ_EChemPET_Electrode_Particle_Layer_DsTabular*
- *ABQ_EChemPET_Electrode_Particle_Layer_OCPTabular*
- *ABQ_EChemPET_Electrode_Particle_Layer_SwellingTabular*
- *ABQ_EChemPET_Electrode_Particle_SEI_CrackFunction_Tabular*
- *ABQ_EChemPET_Electrode_Particle_SEI_DiffTabular*
- *ABQ_EChemPET_Electrode_Particle_SEI_ElecCondTabular*
- *ABQ_EChemPET_Electrode_Particle_SEI_IonCondTabular*
- *ABQ_EChemPET_Electrode_Particle_SEI_OCPTabular*
- *ABQ_EChemPET_Electrolyte_ChargeDischarge*
- *ABQ_EChemPET_Electrolyte_DiffTabular*
- *ABQ_EChemPET_Electrolyte_ElecCond_Tabular*
- *ABQ_EChemPET_Electrolyte_MolarActivityCoeff_fPM*
- *ABQ_EChemPET_Electrolyte_Transference*
- *ABQ_EChemPET_LogScale_Tabular*
- *ABQ_EChemSET_CurrXchgDens_Tabular*
- *ABQ_EChemSET_dUdTEntropy_Tabular*
- *ABQ_EChemSET_Electrode_DiffTabular*
- *ABQ_EChemSET_Electrode_ElecCond_Tabular*
- *ABQ_EChemSET_Electrolyte_DiffTabular*
- *ABQ_EChemSET_Electrolyte_ElecCond_Tabular*
- *ABQ_EChemSET_OCPTabular*

ABQ_EChem_LMB_CurrXchgDens_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This property table type is used in modeling the solid electrodes in lithium metal batteries to specify the current exchange density to use in the Butler-Volmer current density computation.

You can include a property table of type `"ABQ_EChem_LMB_CurrXchgDens_Tabular"` in the table definition to specify the current exchange density to use in the Butler-Volmer current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChem_LMB_CurrXchgDens_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density Exchange current density to use for the Butler-Volmer current density computation in the particle, $\overline{I_0}$.

Independent Variables

Temperature Temperature, θ.

ABQ_EChem_LMB_dUdTEntropy_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This property table type is used in modeling the electrochemical process in a solid electrode battery to specify the change in the open circuit potential with respect to temperature.

You can include a property table of type `"ABQ_EChem_LMB_dUdTEntropy_Tabular"` in the table collection definition to specify the change in the open circuit potential with respect to temperature.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChem_LMB_dUdTEntropy_Tabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

dUdT for Entropy
Derivative of open circuit potential with respect to temperature, $\frac{\partial U}{\partial T}$.

Independent Variables

Temperature
Temperature, θ.

ABQ_EChem_LMB_OCPTabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrodes in Lithium Metal Batteries*

Overview

This property table type is used in modeling the electrochemical process in a solid electrode battery to specify the open circuit potential (OCP) of the solid electrode.

You can include a property table of type `"ABQ_EChem_LMB_OCPTabular"` in the material definition to specify the open circuit potential (OCP) of the solid electrode.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChem_LMB_OCPTabular", PROPERTIES=1,
INDEPENDENT VARIABLES=1
```

Properties

Open Circuit Potential

Open circuit potential of the solid electrode, E_{ref}.

Independent Variables

Temperature

Temperature, θ.

ABQ_EChemPET_Electrode_ElecCond_Tabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the electrical conductivity of the solid phase of the electrode.

You can include a property table of type `"ABQ_EChemPET_Electrode_ElecCond_Tabular"` in the material definition to specify the electrical conductivity of the solid phase of the electrode.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrode_ElecCond_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Conductivity — Electrical conductivity of the electrode, K_s.

Independent Variables

Normalized Particle Concentration — Normalized concentration in the particle, $\widehat{C}_{s,p}$.

ABQ_EChemPET_Electrode_Particle_CurrXchgDens_CrackSEI_Tabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This optional property table type is used in modeling the SEI reformation aging mechanism in batteries to specify the current exchange density to use in the SEI reformation current density computation.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_CurrXchgDens_CrackSEI_Tabular"` in the table definition to specify the current exchange density to use in the SEI reformation current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_CrackSEI_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density Exchange current density to use for the SEI reformation aging current density computation in the particle, I_{0,SEI_reform}.

Independent Variables

Normalized particle surface concentration Ethylene carbonate concentration, C^S_{EC}.

ABQ_EChemPET_Electrode_Particle_CurrXchgDens_ElecDiss_Tabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the electrode dissolution aging mechanism in batteries to specify the current exchange density to use in the electrode dissolution current density computation.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_CurrXchgDens_ElecDiss_Tabular"` in the table definition to specify the current exchange density to use in the electrode dissolution current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_ElecDiss_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density Exchange current density to use for the electrode dissolution current density computation in the particle, $I_{0,Diss}$.

Independent Variables

Temperature Temperature, θ.

ABQ_EChemPET_Electrode_Particle_CurrXchgDens_LPL_Tabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the lithium plating aging mechanism in batteries to specify the current exchange density to use in the lithium plating current density computation.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_CurrXchgDens_LPL_Tabular"` in the table definition to specify the current exchange density to use in the lithium plating current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_LPL_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density	Exchange current density to use for the lithium plating current density computation in the particle, $I_{0,LPL}$.

Independent Variables

Plated lithium concentration	Plated lithium concentration, C_{LPL}.

ABQ_EChemPET_Electrode_Particle_CurrXchgDens_SEI_Tabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the SEI aging mechanism in batteries to specify the current exchange density to use in the SEI current density computation.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_CurrXchgDens_SEI_Tabular"` in the table definition to specify the current exchange density to use in the SEI current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_SEI_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density — Exchange current density to use for the SEI current density computation in the particle, $I_{0,SEI}$.

Independent Variables

Concentration of ethylene carbonate — Ethylene carbonate concentration on surface of particle, C_{ec}^{S}.

ABQ_EChemPET_Electrode_Particle_CurrXchgDens_Tabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the current exchange density to use in the Butler-Volmer current density computation.

You can include a property table of type "ABQ_EChemPET_Electrode_Particle_CurrXchgDens_Tabular" in the material definition to specify the current exchange density to use in the Butler-Volmer current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_CurrXchgDens_Tabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

Current Exchange Density Exchange current density to use for the Butler-Volmer current density computation in the particle, $\overline{I_0}$.

Independent Variables

Normalized Particle Surface Concentration Normalized concentration on the surface of the particle, $\hat{C}_{s,P}^{\text{surf}}$.

ABQ_EChemPET_Electrode_Particle_dUdTEntropy_Tabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the change in open circuit potential with respect to temperature.

You can include a property table of type "ABQ_EChemPET_Electrode_Particle_dUdTEntropy_Tabular" in the material definition to specify the change in open circuit potential with respect to temperature.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_dUdTEntropy_Tabular", PROPERTIES=1,
INDEPENDENT VARIABLES=1
```

Properties

dUdT for Entropy

Derivative of open circuit potential with respect to temperature, $\frac{\partial U}{\partial T}$.

Independent Variables

Normalized Particle Concentration

Normalized concentration in the particle, $\widehat{C}_{s,p}$.

ABQ_EChemPET_Electrode_Particle_ElecDiss_OCPTabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the aging mechanism in a battery to specify the open circuit potential (OCP) of the electrode dissolution of the particle.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_ElecDiss_OCPTabular"` in the material definition to specify the open circuit potential (OCP) of the electrode dissolution of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_ElecDiss_OCPTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Open Circuit Potential — Open circuit potential of the microscale particle, E_{diss}.

Independent Variables

Temperature — Temperature, θ.

ABQ_EChemPET_Electrode_Particle_Layer_DsTabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the diffusion coefficient of the microscale particle.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_Layer_DsTabular"` in the material definition to specify the diffusion coefficient of the microscale particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrode_Particle_Layer_DsTabular",
 PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Particle Diffusion Coefficient — Diffusion coefficient of the microscale particle, D_s.

Independent Variables

Normalized Particle Concentration — Normalized concentration in the particle, $\widehat{C}_{s,P}$.

ABQ_EChemPET_Electrode_Particle_Layer_OCPTabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the open circuit potential (OCP) of the particle.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_Layer_OCPTabular"` in the material definition to specify the open circuit potential (OCP) of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_OCPTabular", PROPERTIES=1,
INDEPENDENT VARIABLES=1
```

Properties

Open Circuit Potential — Open circuit potential of the microscale particle, E_{ref}.

Independent Variables

Normalized Surface Concentration — Normalized concentration of ions on the surface of the microscale particle, $\hat{C}_{s,P}^{\text{surf}}$.

ABQ_EChemPET_Electrode_Particle_Layer_SwellingTabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the change in volume of the particle during the lithiation and delithiation processes.

You can include a property table of type ABQ_EChemPET_Electrode_Particle_Layer_SwellingTabular in the material definition to specify the change in volume of the particle during the lithiation and delithiation processes.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_SwellingTabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

Volumetric Ratio Ratio of current volume at specified normalized concentration to the reference volume. This ratio is used to determine the volume fraction change during lithiation or delithiation, $\frac{V_p}{V_p^0}$.

Independent Variables

Normalized Particle Concentration Normalized concentration in the particle, $\widehat{C}_{s,p}$.

ABQ_EChemPET_Electrode_Particle_SEI_CrackFunction_Tabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This optional property table type is used in modeling the SEI reformation aging mechanism in batteries to specify the cracking function to be used in the SEI reformation current density computation.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_SEI_CrackFunction_Tabular"` in the table definition to specify the cracking function to be used in the current exchange density to use in the SEI reformation current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_SEI_CrackFunction_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Crack function Crack function to be used for the SEI reformation aging current density computation in the particle, f_{crack}.

Independent Variables

Normalized average particle concentration

Normalized average concentration of the particle, $\widehat{C_{s,P}}$.

ABQ_EChemPET_Electrode_Particle_SEI_DiffTabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the aging mechanism in a battery to specify the diffusivity of the SEI of the particle.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_SEI_DiffTabular"` in the material definition to specify the diffusivity of the SEI of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_SEI_DiffTabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

Diffusivity — Diffusivity of the SEI, D_{EC}.

Independent Variables

Concentration — Ethylene carbonate concentration, C_{EC}^{S}.

ABQ_EChemPET_Electrode_Particle_SEI_ElecCondTabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the aging mechanism in a battery to specify the electrical conductivity of the SEI of the particle.

You can include a property table of type "ABQ_EChemPET_Electrode_Particle_SEI_ElecCondTabular" in the material definition to specify the electrical conductivity of the SEI of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_SEI_ElecCondTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Ion conductivity Electrical conductivity of the SEI, σ_{SEI}.

Independent Variables

Concentration Ethylene carbonate concentration, C^{S}_{EC}.

ABQ_EChemPET_Electrode_Particle_SEI_IonCondTabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the aging mechanism in a battery to specify the ionic conductivity of the SEI of the particle.

You can include a property table of type "ABQ_EChemPET_Electrode_Particle_SEI_IonCondTabular" in the material definition to specify the ionic conductivity of theSEI of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_SEI_IonCondTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Ion conductivity — Ion conductivity of the SEI, κ_{SEI}.

Independent Variables

Concentration — Ethylene carbonate concentration, C^{S}_{EC}.

ABQ_EChemPET_Electrode_Particle_SEI_OCPTabular

Products: Abaqus/Standard

References:

- *Modeling Aging in Batteries*

Overview

This property table type is used in modeling the aging mechanism in a battery to specify the open circuit potential (OCP) of the SEI of the particle.

You can include a property table of type `"ABQ_EChemPET_Electrode_Particle_SEI_OCPTabular"` in the material definition to specify the open circuit potential (OCP) of the SEI of the particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE,
NAME="ABQ_EChemPET_Electrode_Particle_Layer_SEI_OCPTabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

Open Circuit Potential — Open circuit potential of the microscale particle, E_{SEI}.

Independent Variables

Concentration — Ethylene carbonate concentration, C_{EC}^{S}.

ABQ_EChemPET_Electrolyte_ChargeDischarge

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*
- *Modeling Aging in Batteries*

Overview

This property table type is optionally used in modeling the electrochemical process in a liquid electrolyte battery to specify a table of values that identify whether a given step in an analysis is a charging or discharging step. If this table is defined, the value interpolated from the table based on the step information is passed into user subroutines UECHEMECDBV, UECHEMECDSEI, UECHEMECDSEIREFORM, UECHEMECDLPL, UECHEMECDDISS, and UEREF.

You can optionally include a property table of type "ABQ_EChemPET_Electrolyte_ChargeDischarge" in the material definition to specify an integer value that will help you identify whether the current step is a charge or discharge step for the user subroutine–based intercalation current definitions in a battery analysis.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChempET_Electrolyte_ChargeDischarge",
PROPERTIES=1, INDEPENDENT VARIABLES=2
```

Properties

ChargeDischarge A value that is passed during the analysis to the user subroutines that you can use to interpret whether the battery is undergoing a charge or discharge during the current time and step.

Independent Variables

Step Time Step time value.

Step Number Step number.

ABQ_EChemPET_Electrolyte_DiffTabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the diffusivity of ions in the electrolyte.

You can include a property table of type `"ABQ_EChemPET_Electrolyte_DiffTabular"` in the material definition to specify the diffusivity of ions in the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte_DiffTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=2
```

Properties

Ion Diffusivity
Ion diffusivity of the electrolyte, D_e.

Independent Variables

Ion Concentration
Ion concentration in the electrolyte, C_e.

Volume Fraction of Liquid Phase
Volume fraction of the electrolyte in the electrode, ε_e.

ABQ_EChemPET_Electrolyte_ElecCond_Tabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the ionic conductivity of the electrolyte.

You can include a property table of type `"ABQ_EChemPET_Electrolyte_ElecCond_Tabular"` in the material definition to specify the ionic conductivity of the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte_ElecCond_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=2
```

Properties

Conductivity	Ionic conductivity of the electrolyte, K_e.

Independent Variables

Ion Concentration	Ion concentration in the electrolyte, C_e.
Volume Fraction of Liquid Phase	Volume fraction of the electrolyte in the electrode, ε_e.

ABQ_EChemPET_Electrolyte_MolarActivityCoeff_fPM

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the molar activity coefficient of the electrolyte.

You can include a property table of type "ABQ_EChemPET_Electrolyte_MolarActivityCoeff_fPM" in the material definition to specify the molar activity coefficient of the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte_MolarActivityCoeff_fPM",
 PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Molar Activity Coefficient	Molar activity coefficient that accounts for deviations from the ideal behavior in a mixture of chemical substances, $\frac{\mathrm{d}\ln f_{\pm}}{\mathrm{d}\ln C_e}$.

Independent Variables

Ion Concentration	Ion concentration in the electrolyte, C_e.

ABQ_EChemPET_Electrolyte_Transference

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the transference number of the electrolyte.

You can include a property table of type `"ABQ_EChemPET_Electrolyte_Transference"` in the material definition to specify the transference number of the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_Electrolyte_Transference",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Transference Number Transference number that defines the fraction of the total electrical current that ions carry in the electrolyte, t_+.

Independent Variables

Ion Concentration Ion concentration in the electrolyte, C_e.

ABQ_EChemPET_LogScale_Tabular

Products: Abaqus/Standard

References:

- *Coupled Thermal-Electrochemical Analysis*

Overview

This property table type is used in modeling the electrochemical process in a battery to specify the logarithmic value of the diffusion coefficient of the microscale particle.

You can include a property table of type `"ABQ_EChemPET_LogScale_Tabular"` in the material definition to specify the logarithmic value ($\mathbf{Log}_{10}$) of the diffusion coefficient of the microscale particle.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemPET_LogScale_Tabular", PROPERTIES=1,
INDEPENDENT VARIABLES=1
```

Properties

Logarithmic Value of Particle Diffusion Coefficient	$\mathbf{Log}_{10}$ of the particle diffusion coefficient of the microscale particle, $\mathbf{Log}_{10}\, D_s$.

Independent Variables

Normalized Particle Concentration	Normalized concentration in the particle, $\widehat{C}_{s,p}$.

ABQ_EChemSET_CurrXchgDens_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid electrolyte interface in solid state batteries to specify the current exchange density to use in the Butler-Volmer current density computation.

You can include a property table of type `"ABQ_EChemSET_CurrXchgDens_Tabular"` in the table definition to specify the current exchange density to use in the Butler-Volmer current density computation.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_CurrXchgDens_Tabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=2
```

Properties

Current Exchange Density — Exchange current density to use for the Butler-Volmer current density computation at the interface, $\overline{I_0}$.

Independent Variables

Ion Concentration — Ion concentration in the separator, C_e.

Species Concentration — Species concentration in the electrode, C_s.

ABQ_EChemSET_dUdTEntropy_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid electrolyte interface in a solid state battery to specify the change in the open circuit potential with respect to temperature.

You can include a property table of type `"AABQ_EChemSET_dUdTEntropy_Tabular"` in the table collection definition to specify the change in the open circuit potential with respect to temperature.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_dUdTEntropy_Tabular", PROPERTIES=1,
 INDEPENDENT VARIABLES=1
```

Properties

dUdT for Entropy

Derivative of open circuit potential with respect to temperature, $\frac{\partial U}{\partial T}$.

Independent Variables

Temperature

Temperature, θ.

ABQ_EChemSET_Electrode_DiffTabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid state battery to specify the diffusivity of species in the electrode.

You can include a property table of type `"ABQ_EChemSET_Electrolyte_DiffTabular"` in the material definition to specify the diffusivity of species in the electrode.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_Electrode_DiffTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Species Diffusivity — Species diffusivity in the electrode, D_s.

Independent Variables

Ion Concentration — Species concentration in the electrode, C_s.

ABQ_EChemSET_Electrode_ElecCond_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the electrochemical process in a solid state battery to specify the electrical conductivity of the solid phase of the electrode.

You can include a property table of type `"ABQ_EChemSET_Electrode_ElecCond_Tabular"` in the material definition to specify the electrical conductivity of the solid phase of the electrode.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_Electrode_ElecCond_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Conductivity Electrical conductivity of the solid, K_s.

Independent Variables

Species Concentration Species concentration, C_s.

ABQ_EChemSET_Electrolyte_DiffTabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid electrolyte in a battery to specify the diffusivity of ions in the electrolyte.

You can include a property table of type `"ABQ_EChemSET_Electrolyte_DiffTabular"` in the material definition to specify the diffusivity of ions in the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_Electrolyte_DiffTabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Ion Diffusivity — Ion diffusivity of the electrolyte, D_e.

Independent Variables

Ion Concentration — Ion concentration in the electrolyte, C_e.

ABQ_EChemSET_Electrolyte_ElecCond_Tabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid electrolyte in a battery to specify the electrical conductivity of the electrolyte.

You can include a property table of type `"ABQ_EChemSET_Electrolyte_ElecCond_Tabular"` in the material definition to specify the electrical conductivity of the electrolyte.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_Electrolyte_ElecCond_Tabular",
PROPERTIES=1, INDEPENDENT VARIABLES=1
```

Properties

Conductivity — Electrical conductivity of the electrolyte, K_e.

Independent Variables

Ion Concentration — Ion concentration in the electrolyte, C_e.

ABQ_EChemSET_OCPTabular

Products: Abaqus/Standard

References:

- *Modeling Solid Electrolytes and Solid-State Batteries*

Overview

This property table type is used in modeling the solid electrolyte interface in a solid state battery to specify the open circuit potential (OCP) of the solid electrode.

You can include a property table of type `"ABQ_EChemSET_OCPTabular"` in the material definition to specify the open circuit potential (OCP) of the solid electrode.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_EChemSET_OCPTabular", PROPERTIES=1,
INDEPENDENT VARIABLES=1
```

Properties

Open Circuit Potential Open circuit potential of the solid electrode, E_{ref}.

Independent Variables

Species Concentration concentration, C_s.

Coupled Pore Fluid Flow and Stress Analysis

In this section:

- *Coupled Pore Fluid Diffusion and Stress Analysis*
- *Undrained Pore Fluid Flow and Stress Analysis*
- *Geostatic Stress State*

Coupled Pore Fluid Diffusion and Stress Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A coupled pore fluid diffusion/stress analysis:

- is used to model single phase, partially or fully saturated fluid flow through porous media;
- can be performed in terms of either total pore pressure or excess pore pressure by including or excluding the pore fluid weight;
- requires the use of pore pressure elements with associated pore fluid flow properties defined;
- can, optionally, also model heat transfer due to conduction in the soil skeleton and the pore fluid, and convection due to the flow of the pore fluid, through the use of coupled temperature–pore pressure displacement elements;
- can be transient or steady-state;
- can be linear or nonlinear; and
- can include pore pressure contact between bodies (see *Pore Fluid Contact Properties*).

References:

- *Defining an Analysis*
- *About Pore Fluid Flow Properties*
- **SOILS*
- *Defining pore fluid expansion*
- *Configuring an effective stress analysis for fluid-filled porous media*

Typical Applications

Some of the more common coupled pore fluid diffusion/stress (and, optionally, thermal) analysis problems that can be analyzed with Abaqus/Standard are:

Saturated flow

Soil mechanics problems generally involve fully saturated flow, since the solid is fully saturated with ground water. Typical examples of saturated flow include consolidation of soils under foundations and excavation of tunnels in saturated soil.

Partially saturated flow

Partially saturated flow occurs when the wetting liquid is absorbed into or exsorbed from the medium by capillary action. Irrigation and hydrology problems generally include partially saturated flow.

Combined flow

Combined fully saturated and partially saturated flow occurs in problems such as seepage of water through an earth dam, where the position of the phreatic surface (the boundary between fully saturated and partially saturated soil) is of interest.

Moisture migration

Although not normally associated with soil mechanics, moisture migration problems can also be solved using the coupled pore fluid diffusion/stress procedure. These problems might involve partially saturated flow in polymeric materials such as paper towels and sponge-like materials; in the biomedical industry they might also involve saturated flow in hydrated soft tissues.

Combined heat transfer and pore fluid flow

In some applications, such as a source of heat buried in soil, it is important to model the coupling between the mechanical deformation, pore fluid flow, and heat transfer. In such problems the difference in the thermal expansion coefficients between the soil and the pore fluid often plays an important role in determining the rate of diffusion of the pore fluid and heat from the source.

Flow through Porous Media

A porous medium is modeled in Abaqus/Standard by a conventional approach that considers the medium as a multiphase material and adopts an effective stress principle to describe its behavior. The porous medium modeling provided considers the presence of two fluids in the medium. One is the "wetting liquid," which is assumed to be relatively (but not entirely) incompressible. Often the other is a gas, which is relatively compressible. An example of such a system is soil containing ground water. When the medium is partially saturated, both fluids exist at a point; when it is fully saturated, the voids are completely filled with the wetting liquid. The elementary volume, dV, is made up of a volume of grains of solid material, dV_g; a volume of voids, dV_v; and a volume of wetting liquid, $dV_w \leq dV_v$, that is free to move through the medium if driven. In some systems (for example, systems containing particles that absorb the wetting liquid and swell in the process) there might also be a significant volume of trapped wetting liquid, dV_t.

The porous medium is modeled by attaching the finite element mesh to the solid phase; fluid can flow through this mesh. The mechanical part of the model is based on the effective stress principle defined in *Effective stress principle for porous media*.

The model also uses a continuity equation for the mass of wetting fluid in a unit volume of the medium. This equation is described in *Continuity statement for the wetting liquid phase in a porous medium*. It is written with pore pressure (the average pressure in the wetting fluid at a point in the porous medium) as the basic variable (degree of freedom 8 at the nodes). The conjugate flux variable is the volumetric flow rate at the node, $\mathbf{f}$. The porous medium is partially saturated when the pore liquid pressure, u_w, is negative.

Coupled Flow and Heat Transfer through Porous Media

Optionally, heat transfer due to conduction in the soil skeleton and pore fluid, as well as convection in the pore fluid, can also be modeled. This capability represents an enhancement to the basic pore fluid flow capabilities discussed in the earlier paragraphs and requires the use of coupled temperature–pore pressure elements that have temperature as an additional degree of freedom (degree of freedom 11 at the nodes) in addition to the pore pressure and the displacement components. When you use the coupled temperature–pore pressure elements, Abaqus solves the heat transfer equation in addition to and in a fully coupled manner with the continuity equation and the mechanical equilibrium equations. Only linear brick, first-order axisymmetric, and second-order modified tetrahedrons are available for modeling coupled heat transfer with pore fluid flow and mechanical deformation.

Total and Excess Pore Fluid Pressure

The coupled pore fluid diffusion/stress analysis capability can provide solutions either in terms of total or "excess" pore fluid pressure. The excess pore fluid pressure at a point is the pore fluid pressure in excess of the hydrostatic pressure required to support the weight of pore fluid above the elevation of the material point. The difference between total and excess pore pressure is relevant only for cases in which gravitational loading is important; for example, when the loading provided by the hydrostatic pressure in the pore fluid is large or when effects like "wicking" (transient capillary suction of liquid into a dry column) are being studied. Total pore pressure solutions are provided when the gravity distributed load is used to define the gravity load on the model. Excess pore pressure solutions are provided in all other cases; for example, when gravity loading is defined with body force distributed loads.

Steady-State Analysis

Steady-state coupled pore pressure/effective stress analysis assumes that there are no transient effects in the wetting liquid continuity equation; that is, the steady-state solution corresponds to constant wetting liquid velocities and constant volume of wetting liquid per unit volume in the continuum. Thus, for example, thermal expansion of the liquid phase has no effect on the steady-state solution: it is a transient effect. Therefore, the time scale chosen during steady-state analysis is relevant only to rate effects in the constitutive model used for the porous medium (excluding creep and viscoelasticity, which are disabled in steady-state analysis).

Mechanical loads and boundary conditions can be changed gradually over the step by referring to an amplitude curve to accommodate possible geometric nonlinearities in the response.

The steady-state coupled equations are strongly unsymmetric; therefore, the unsymmetric matrix solution and storage scheme is used automatically for steady-state analysis steps (see *Defining an Analysis*).

If heat transfer is modeled using the coupled temperature–pore pressure elements, the steady-state solution neglects all transient effects in the heat transfer equation and provides only the steady-state temperature distribution.

Input File Usage: **SOILS*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Steady state**

Incrementation

You can specify a fixed time increment size in a coupled pore fluid diffusion/stress analysis, or Abaqus/Standard can select the time increment size automatically. Automatic incrementation is recommended because the time increments in a typical diffusion analysis can increase by several orders of magnitude during the simulation. If you do not activate automatic incrementation, fixed time increments will be used.

Input File Usage: Use the following option to activate automatic incrementation in steady-state analysis:

**SOILS*, UTOL=*any arbitrary nonzero value*

The solution does not depend on the value specified for UTOL; this value is simply a flag for automatic incrementation.

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Steady state**; **Incrementation**: **Type: Automatic**

Transient Analysis

In a transient coupled pore pressure/effective stress analysis the backward difference operator is used to integrate the continuity equation and the heat transfer equation (if heat transfer is modeled): this operator provides unconditional stability so that the only concern with respect to time integration is accuracy. You can provide the time increments, or they can be selected automatically.

The coupled partially saturated flow equations are strongly unsymmetric, so the unsymmetric solver is used automatically if you request partially saturated analysis (by including absorption/exsorption behavior in the material definition). The unsymmetric solver is also activated automatically when gravity distributed loading is used during a soils consolidation analysis.

For fully saturated flow analyses in which finite-sliding coupled pore pressure-displacement contact is modeled using contact pairs, certain contributions to the model's stiffness matrix are unsymmetric. Using the unsymmetric solver can sometimes improve convergence in such cases since Abaqus does not automatically do so.

For fully saturated flow analyses in which heat transfer is also modeled, the contributions to the model's stiffness matrix arising from convective heat transfer due to pore fluid flow are unsymmetric. Using the unsymmetric solver can sometimes improve convergence in such cases since Abaqus does not automatically do so.

Spurious Oscillations due to Small Time Increments

The integration procedure used in Abaqus/Standard for consolidation analysis introduces a relationship between the minimum usable time increment and the element size, as shown below for fully saturated and partially saturated flows. If time increments smaller than these values are used, spurious oscillations might appear in the solution (except for partially saturated cases when linear elements or modified triangular elements are used; in these cases Abaqus/Standard uses a special integration scheme for the wetting liquid storage term to avoid the problem). These nonphysical oscillations might cause problems if pressure-sensitive plasticity is used to model the porous medium and might lead to convergence difficulties in partially saturated analyses. If the problem requires analysis with smaller time increments than the relationships given below allow, a finer mesh is required. Generally there is no upper limit on the time step except accuracy, since the integration procedure is unconditionally stable unless nonlinearities cause convergence problems.

Fully Saturated Flow

A simple guideline that can be used for the minimum usable time increment in the case of fully saturated flow is

$$\Delta t > \frac{\gamma_w(1+\beta v_w)}{6E\mathbf{k}}\left(1-\frac{E}{K_g}\right)^2(\Delta\ell)^2,$$

where

Δt

is the time increment,

γ_w

is the specific weight of the wetting liquid,

E

is the Young's modulus of the soil,

$\mathbf{k}$

is the permeability of the soil (see *Permeability*),

v_w

is the magnitude of the velocity of the pore fluid,

β

is the velocity coefficient in Forchheimer's flow law ($\beta = 0$ in the case of Darcy flow),

K_g

is the bulk modulus of the solid grains (see *Porous Bulk Moduli*), and

$\Delta\ell$

is a typical element dimension.

Partially Saturated Flow

In partially saturated flow cases the corresponding guideline for the minimum time increment is

$$\Delta t > \frac{\gamma_w n^0 (1+\beta v_w)}{6 k_s \mathbf{k}} \frac{ds}{du_w} (\Delta\ell)^2,$$

where

s

is the saturation;

k_s

is the permeability-saturation relationship;

ds/du_w

is the rate of change of saturation with respect to pore pressure (see *Sorption*);

n^0

is the initial porosity of the material; and the other parameters are as defined for the case of fully saturated flow.

Including a Stabilization Term to Eliminate Spurious Oscillations

For materials with very low permeability, a highly refined mesh with very small element lengths is required to satisfy the minimum time increment in the above equations. However, such time increment and mesh size requirements might not be feasible for practical problems. Competing time increment requirements in a fracture analysis are another complication.

The spurious oscillations are due to equal-order approximations for the pore pressure and displacement equations in the finite element analysis. Pressure projections into the strain space can be considered to eliminate the approximation inconsistency. A local stabilized method involving polynomial projections over individual elements is implemented. In this stabilization approach that follows *Dohrmann and Bochev*, a stabilization term that penalizes pressure deviations from the consistent polynomial order supplements the local pressure projections.

When you apply stabilization, you should ensure that the stabilized viscous fluid volume fluxes are relatively small compared with the applied fluxes and the reaction fluid volume fluxes due to prescribed pressure in the model.

Input File Usage: **PERMEABILITY*, STABILIZE=*stabilization_coeff_value*

Abaqus/CAE Usage: Specifying a stabilization coefficient is not supported in Abaqus/CAE.

Fixed Incrementation

If you choose fixed time incrementation, fixed time increments equal to the size of the user-specified initial time increment, Δt_0, will be used. Fixed incrementation is not generally recommended because the time increments in a typical diffusion analysis can increase over several orders of magnitude during the simulation; automatic incrementation is usually a better choice.

Input File Usage: **SOILS*, CONSOLIDATION
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **Type: Fixed**, **Increment size:** Δt_0

Automatic Incrementation

If you choose automatic time incrementation, you must specify two (three if heat transfer is also modeled) tolerance parameters.

The accuracy of the time integration of the flow continuity equations is governed by the maximum wetting liquid pore pressure change, Δu_w^{max}, allowed in an increment. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment in the analysis.

If heat transfer is modeled, the accuracy of time integration is also governed by the maximum temperature change, $\Delta\theta_{max}$, allowed in an increment. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis.

The accuracy of the integration of the time-dependent (creep) material behavior is governed by the maximum strain rate change allowed at any point during an increment, $errtol \geq \left(\dot{\bar{\varepsilon}}^{cr}{}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}{}_{t}\right)\Delta t$, as described in *Rate-Dependent Plasticity: Creep and Swelling*.

Input File Usage: If heat transfer is not modeled:

**SOILS*, CONSOLIDATION, UTOL=Δu_w^{max}, , CETOL=*errtol*

If heat transfer is modeled:

**SOILS*, CONSOLIDATION, UTOL=Δu_w^{max}, DELTMX=$\Delta\theta_{max}$, CETOL=*errtol*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **Type: Automatic**, **Max. pore pressure change per increment:** Δu_w^{max}, **Creep/swelling/viscoelastic strain error tolerance:** *errtol*

Specifying the maximum temperature change per increment is not supported in Abaqus/CAE.

Ending a Transient Analysis

Transient soils analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or the time period ends, whichever comes first. When heat transfer is not modeled, steady state is defined by a maximum permitted rate of change of pore pressure with time: when all pore pressures are changing at less than the user-defined rate, the analysis terminates. However, with heat transfer included, the analysis terminates only when both the pore pressure and temperature are changing at less than the user-defined rates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

**SOILS*, CONSOLIDATION, END=PERIOD (default)

Use the following option to end the analysis based on the pore pressure and, if heat transfer is modeled, temperature change rate:

**SOILS*, CONSOLIDATION, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **End step when pore pressure change rate is less than**

If heat transfer is modeled, directly specifying the temperature change rate to define steady state is not supported in Abaqus/CAE.

Neglecting Creep during a Transient Analysis

You can specify that creep or viscoelastic response should be neglected during a consolidation analysis, even if creep or viscoelastic material properties have been defined.

Input File Usage: **SOILS*, CONSOLIDATION, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable Problems

Some types of analyses might develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it might not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in *Automatic Stabilization of Unstable Problems*.

Optional Modeling of Coupled Heat Transfer

When coupled temperature–pore pressure elements are used, heat transfer is modeled in these elements by default. However, you might optionally choose to switch off heat transfer within these elements during some steps in the analysis. This feature might be helpful in reducing computation time during certain phases in the analysis when heat transfer is not an important part of the overall physics of the problem.

Input File Usage: Use the following option either during a transient or a steady-state procedure to suppress heat transfer modeling:

**SOILS*, CONSOLIDATION, HEAT=NO

Abaqus/CAE Usage: Switching off the heat transfer part of the physics is not supported in Abaqus/CAE.

Units

In coupled problems where two or more different fields are being solved, you must be careful when choosing the units of the problem. If the choice of units is such that the numbers generated by the equations for the different fields differ by many orders of magnitude, the precision on some computers might be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid badly conditioned matrices. For example, consider using units of Mpascal instead of pascal for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations and the pore flow continuity equations.

Initial Conditions

Initial conditions can be applied as defined in *Initial Conditions*.

The initial effective stress/pore pressure conditions defined for an element are assumed to be acting on the initial configuration of the element. If the initial effective stress/pore pressure conditions are removed during the step, the element returns to a stress-free configuration that is different from the initial one. Since displacements and total strain output are measured relative to the initial configuration, the stress-free configuration will have nonzero values for the displacement and total strain fields that will depend on the initial conditions. While it is easy to verify the above behavior analytically in a one-element problem subjected to an initial stress and pore pressure field, the situation in a complex boundary value problem is determined by other factors that might make it difficult to resolve analytically.

Defining Initial Pore Fluid Pressures

Initial values of pore fluid pressures, u_w, can be defined at the nodes.

Input File Usage: **INITIAL CONDITIONS*, TYPE=PORE PRESSURE

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Pore pressure** for the **Types for Selected Step**

Defining Initial Void Ratios

Initial values of the void ratio, e, can be given at the nodes. The void ratio is defined as the ratio of the volume of voids to the volume of solid material (see *Effective stress principle for porous media*). The evolution of void ratio is governed by the deformation of the different phases of the material, as discussed in detail in *Constitutive behavior in a porous medium*.

Input File Usage: **INITIAL CONDITIONS*, TYPE=RATIO

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Void ratio** for the **Types for Selected Step**

Defining Initial Saturation

Initial saturation values, s, can be given at the nodes. Saturation is defined as the ratio of wetting fluid volume to void volume (see *Effective stress principle for porous media*).

Input File Usage: **INITIAL CONDITIONS*, TYPE=SATURATION

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Saturation** for the **Types for Selected Step**

Defining Initial Stresses

An initial (effective) stress field can be specified (see *Initial Conditions*).

Most geotechnical problems begin from a geostatic state, which is a steady-state equilibrium configuration of the undisturbed soil or rock body under geostatic loading and usually includes both horizontal and vertical components. It is important to establish these initial conditions correctly so that the problem begins from an equilibrium state. The geostatic procedure can be used to verify that the user-defined initial stresses are indeed in equilibrium with the given geostatic loads and boundary conditions (see *Geostatic Stress State*).

Input File Usage: Use one of the following options:

**INITIAL CONDITIONS*, TYPE=STRESS
**INITIAL CONDITIONS*, TYPE=STRESS, GEOSTATIC

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Mechanical** for the **Category** and **Stress** or **Geostatic stress** for the **Types for Selected Step**

Defining Initial Temperature

Initial temperature values can be defined at the nodes.

Input File Usage: **INITIAL CONDITIONS*, TYPE=TEMPERATURE

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Temperature** for the **Types for Selected Step**

Boundary Conditions

Boundary conditions can be applied to displacement degrees of freedom 1–6 and to pore pressure degree of freedom 8 (*Boundary Conditions*). In addition, boundary conditions can also be applied to temperature degree of freedom 11 if heat transfer is modeled using coupled temperature–pore pressure elements. During the analysis prescribed boundary conditions can be varied by referring to an amplitude curve (*Amplitude Curves*). If no amplitude reference is given, the default variation of a boundary condition in a coupled pore fluid diffusion/stress analysis step is as defined in *Defining an Analysis*.

If the pore pressure is prescribed with a boundary condition, fluid is assumed to enter and leave through the node as needed to maintain the prescribed pressure. Likewise, if the temperature is prescribed with a boundary condition, heat is assumed to enter and leave through the node as needed to maintain the prescribed temperature.

Loads

The following loading types can be prescribed in a coupled pore fluid diffusion/stress analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*. The magnitude and direction of gravitational loading are usually defined by using the gravity distributed load type.
- Pore fluid flow is controlled as described in *Pore Fluid Flow*.

If heat transfer is modeled, the following types of thermal loading can also be prescribed (*Thermal Loads*). These loads are not supported in Abaqus/CAE during a coupled thermal pore pressure/stress analysis.

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Predefined Fields

The following predefined fields can be prescribed, as described in *Predefined Fields*:

- For a coupled pore fluid diffusion/stress analysis that does not model heat transfer and uses regular pore pressure elements, temperature is not a degree of freedom and nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- Predefined temperature fields are not allowed in coupled pore fluid diffusion/stress analysis that also models heat transfer. Boundary conditions should be used instead to specify temperatures, as described earlier.
- The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any.

Material Options

Any of the mechanical constitutive models available in Abaqus/Standard can be used to model the porous material.

In problems formulated in terms of total pore pressure, you must include the density of the dry material in the material definition (see *Density*).

You can use a permeability material property to define the specific weight of the wetting liquid, γ_w; the permeability, $\mathbf{k}$, and its dependence on the void ratio, e, and saturation, k_s; and the flow velocity, β (see *Permeability*).

You can define the compressibility of the solid grains and of the permeating fluid in both fully and partially saturated flow problems (see *Elastic Behavior of Porous Materials*). If you do not specify the porous bulk moduli, the materials are assumed to be fully incompressible.

For partially saturated flow you must define the porous medium's absorption/exsorption behavior (see *Sorption*).

Gel swelling (*Swelling Gel*) and volumetric moisture swelling of the solid skeleton (*Moisture Swelling*) can be included in partially saturated cases. These effects are usually associated with modeling of moisture migration in polymeric systems rather than with geotechnical systems.

Thermal Properties if Heat Transfer is Modeled

In problems that model heat transfer, the thermal conductivity for either the solid material or the permeating fluid, or more commonly for both phases, must be defined. Only isotropic conductivity can be specified for the pore fluid. The specific heat, dry material density, and the density of permeating fluid must also be defined for transient heat transfer problems. Latent heat for the phases can be defined if changes in internal energy due to phase changes are important. See *About Thermal Properties* for details on defining thermal properties in Abaqus. Examples of problems that model fully coupled heat transfer along with pore fluid diffusion and mechanical deformation can be found in *Consolidation around a cylindrical heat source* and *Permafrost thawing–pipeline interaction*.

The thermal properties can be defined separately for the solid material and the permeating fluid.

Input File Usage: To define the conductivity, specific heat, density, and latent heat of the permeating fluid, use the following options:

**CONDUCTIVITY*, TYPE=ISO, PORE FLUID
**SPECIFIC HEAT*, PORE FLUID
**LATENT HEAT*, PORE FLUID
**DENSITY*, PORE FLUID

To define the conductivity, specific heat, latent heat of the solid material, and dry material density, use the following options:

**CONDUCTIVITY*, TYPE=ISO or ORTHO or ANISO
**SPECIFIC HEAT*
**LATENT HEAT*
**DENSITY*

Abaqus/CAE Usage: Defining the thermal properties and the density of the permeating fluid is not supported in Abaqus/CAE.

To define the conductivity, specific heat, density, and latent heat of the solid material, use the following options:

Property module: material editor:
Thermal->Conductivity: **Type**: **Isotropic**
Thermal->Specific Heat
General->Density
Thermal->Latent Heat

Thermal Expansion

Thermal expansion can be defined separately for the solid material and for the permeating fluid. In such a case you should repeat the expansion material property, with the necessary parameters, to define the different thermal expansion effects (see *Thermal Expansion*). Thermal expansion will be active only in a consolidation (transient) analysis.

Input File Usage: To define the thermal expansion of the permeating fluid:

*EXPANSION, TYPE=ISO, PORE FLUID

To define the thermal expansion of the solid material:

*EXPANSION, TYPE=ISO or ORTHO or ANISO

Abaqus/CAE Usage: To define the thermal expansion of the permeating fluid:

Property module: material editor: **Other->Pore Fluid->Pore Fluid Expansion**

To define the thermal expansion of the solid material:

Property module: material editor: **Mechanical->Expansion**

Elements

The analysis of flow through porous media in Abaqus/Standard is available for plane strain, axisymmetric, and three-dimensional problems. The modeling of coupled heat transfer effects is available only for plane strain, axisymmetric, and three-dimensional problems. Continuum pore pressure elements are provided for modeling fluid flow through a deforming porous medium in a coupled pore fluid diffusion/stress analysis. These elements have pore pressure degree of freedom 8 in addition to displacement degrees of freedom 1–3. Heat transfer through the porous medium can also be modeled using continuum coupled temperature–pore pressure elements. These elements have temperature degree of freedom 11 in addition to pore pressure degree of freedom 8 and displacement degrees of freedom 1–3. Stress/displacement elements can be used in parts of the model without pore fluid flow. See *Choosing the Appropriate Element for an Analysis Type* for more information.

Output

The element output available for a coupled pore fluid diffusion/stress analysis includes the usual mechanical quantities such as (effective) stress; strain; energies; and the values of state, field, and user-defined variables. In addition, the following quantities associated with pore fluid flow are available:

VOIDR Void ratio, e.

POR Pore pressure, u_w.

SAT Saturation, s.

GELVR Gel volume ratio, n_t.

FLUVR Total fluid volume ratio, n_f.

FLVEL Magnitude and components of the pore fluid effective velocity vector, $\mathbf{f}$.

FLVELM Magnitude, $\mathbf{f}$, of the pore fluid effective velocity vector.

FLVEL*n* Component n of the pore fluid effective velocity vector (n=1, 2, 3).

If heat transfer is modeled, the following element output variables associated with heat transfer are also available:

HFL Magnitude and components of the heat flux vector.

HFL*n* Component *n* of the heat flux vector (*n*=1, 2, 3).

HFLM Magnitude of the heat flux vector.

TEMP Integration point temperatures.

TEMPR Integration point temperature rate.

GRADT Temperature gradient vector.

GRADT*n* Component *n* of the temperature gradient (*n*=1,2,3).

The nodal output available includes the usual mechanical quantities such as displacements, reaction forces, and coordinates. In addition, the following quantities associated with pore fluid flow are available:

CFF Concentrated fluid flow at a node.

POR Pore pressure at a node.

RVF Reaction fluid volume flux due to prescribed pressure. This flux is the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pressure boundary condition. A positive value of RVF indicates that fluid is entering the model.

RVT Reaction total fluid volume (computed only in a transient analysis). This value is the time integrated value of RVF.

If heat transfer is modeled, the following nodal output variables associated with heat transfer are also available:

NT Nodal point temperatures.

RFL Reaction flux values due to prescribed temperature.

RFL*n* Reaction flux value *n* at a node (*n*=11, 12, …).

CFL Concentrated flux values.

CFL*n* Concentrated flux value *n* at a node (*n*=11, 12, …).

All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
…
**********************************
**
```

```
** Material definition
**
**********************************
*MATERIAL, NAME=soil
Data lines to define mechanical properties of the solid material
…
*EXPANSION
Data lines to define the thermal expansion coefficient of the solid grains
*EXPANSION, TYPE=ISO, PORE FLUID
Data lines to define the thermal expansion coefficient of the permeating fluid
*PERMEABILITY, SPECIFIC=γ_w
Data lines to define permeability, k, as a function of the void ratio, e
*PERMEABILITY, TYPE=SATURATION
Data lines to define the dependence of permeability on saturation, k_s(s)
*PERMEABILITY, TYPE=VELOCITY
Data lines to define the velocity coefficient, β(e)
*POROUS BULK MODULI
Data line to define the bulk moduli of the solid grains and the permeating fluid
*SORPTION, TYPE=ABSORPTION
Data lines to define absorption behavior
*SORPTION, TYPE=EXSORPTION
Data lines to define exsorption behavior
*SORPTION, TYPE=SCANNING
Data lines to define scanning behavior (between absorption and exsorption)
*GEL
Data line to define gel behavior in partially saturated flow
*MOISTURE SWELLING
Data lines to define moisture swelling strain as a function of saturation
in partially saturated flow
*CONDUCTIVITY
Data lines to define thermal conductivity of the solid grains if heat transfer is modeled
*CONDUCTIVITY,TYPE=ISO, PORE FLUID
Data lines to define thermal conductivity of the permeating fluid if heat transfer is modeled
*SPECIFIC HEAT
Data lines to define specific heat of the solid grains if transient heat transfer is modeled
*SPECIFIC HEAT,PORE FLUID
Data lines to define specific heat of the permeating fluid if transient heat transfer is modeled
*DENSITY
Data lines to define density of the solid grains if transient heat transfer is modeled
*DENSITY,PORE FLUID
Data lines to define density of the permeating fluid if transient heat transfer is modeled
*LATENT HEAT
Data lines to define latent heat of the solid grains if phase change due to temperature change
 is modeled
*LATENT HEAT,PORE FLUID
Data lines to define latent heat of the permeating fluid if phase change due to temperature change
is modeled
…
**********************************
**
** Boundary conditions and initial conditions
**
**********************************
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC
Data lines to specify initial stresses
```

```
*INITIAL CONDITIONS, TYPE=PORE PRESSURE
Data lines to define initial values of pore fluid pressures
*INITIAL CONDITIONS, TYPE=RATIO
Data lines to define initial values of the void ratio
*INITIAL CONDITIONS, TYPE=SATURATION
Data lines to define initial saturation
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data lines to define initial saturation
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**********************************
**
** Step 1: Optional step to ensure an equilibrium
** geostatic stress field
**
**********************************
*STEP
*GEOSTATIC
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
Data lines to specify mechanical loading
*FLOW and/or *SFLOW and/or *DFLOW and/or *DSFLOW
Data lines to specify pore fluid flow
*CFLUX and/or *DFLUX
Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled
*BOUNDARY
Data lines to specify displacements or pore pressures
*END STEP
**********************************
**
** Step 2: Coupled pore diffusion/stress analysis step
**
**********************************
*STEP (,NLGEOM)
** Use NLGEOM to include geometric nonlinearities
*SOILS
Data line to define incrementation
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to specify mechanical loading
*FLOW and/or *SFLOW and/or *DFLOW and/or *DSFLOW
Data lines to specify pore fluid flow
*CFLUX and/or *DFLUX
Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled
*FILM
Data lines referring to film property table if heat transfer is modeled
*BOUNDARY
Data lines to specify displacements, pore pressures, or temperatures
*END STEP
```

References

- Dohrmann, C., and P. Bochev, “A Stabilized Finite Element Method for the Stokes Problem Based on Polynomial Pressure Projections,” *International Journal for Numerical Methods in Fluids*, vol. 46, pp. 183–201, 2004.

Undrained Pore Fluid Flow and Stress Analysis

Products: Abaqus/Explicit

Overview

An undrained pore fluid flow and stress analysis:

- accounts for the effect of pore fluid pressure in a pure stress analysis to correctly simulate the mechanical response of a granular solid skeleton fully saturated with a permeating fluid;
- is used in situations where the loading rate is significantly faster than the rate at which pore fluid can drain from the porous medium, such that fluid diffusion can be ignored in the analysis;
- can be conducted as part of a dynamic analysis;
- is computationally efficient for the analysis of large models under undrained conditions; and
- requires the specification of the pore fluid density and porous bulk moduli.

References:

- *Defining an Analysis*
- *About Pore Fluid Flow Properties*
- *Specifying material mass density*
- *Defining pore fluid expansion*
- *Defining porous bulk moduli*

Undrained Analysis

A porous medium typically consists of a solid skeleton of numerous solid grains that are "loosely packed" and a pore fluid that flows through a system of interconnected pores among these solid grains. An example of a porous medium is soil containing ground water. When subjected to loading, both the soil and water respond in a manner that leads generally to a fully coupled response where the deformation in the soil influences the pressure of the water and vice versa. In addition, water flows through the porous medium driven by pressure gradients. To simulate the fully coupled response, a fully coupled pore pressure and stress analysis is necessary (see *Coupled Pore Fluid Diffusion and Stress Analysis*).

In some situations when the loading rate is significantly faster than the rate at which pore fluid can drain from the porous medium, the fluid diffusion terms can be ignored. This corresponds to the undrained conditions.

The undrained pore fluid flow and stress analysis functionality in Abaqus/Explicit is intended for the simulation of fully saturated porous media under undrained conditions. The functionality enables modeling the increase in stiffness of the soil in applications such as trenching, gouging, and pile driving. Another typical class of problems that can be modeled is soil liquefaction, such as quicksand, quick clay, turbidity currents, and earthquake-induced liquefaction. In all of these applications, fluid diffusion terms can be ignored when the rate of loading is fast compared to the characteristic time of the diffusion process.

For problems where the fluid diffusion terms are important and fluid can rapidly diffuse, a fully coupled pore pressure and stress analysis must be performed (see *Coupled Pore Fluid Diffusion and Stress Analysis*).

Input File Usage: Use the following option for an undrained pore fluid and stress analysis:

*DYNAMIC, EXPLICIT, PORE PRESSURE=*UNDRAINED*

Abaqus/CAE Usage: Undrained pore fluid flow and stress analysis is not supported in Abaqus/CAE.

Governing Equations

For the coupled pore fluid flow and stress analysis, the continuity equation for the fluid diffusion is given by

$$\left(\frac{n}{K_w}+\frac{(\alpha-n)}{K_g}\right)\frac{du_w}{dt}+\frac{\partial}{\partial \mathbf{x}}\left(\frac{\rho_w}{\rho_w^0}nv_w\right)=-\alpha\dot{\varepsilon}_{vol}+n\dot{\varepsilon}_w^{th}+\left(\alpha-n\right)\dot{\varepsilon}_g^{th},$$

where

u_w
is the fluid pore pressure,

K_w
is the bulk modulus of the fluid,

K_g
is the bulk modulus of the solid grains,

n
is the porosity of the medium,

ρ_w
is the density of the fluid,

ρ_w^0
is the density of the fluid in the reference configuration,

$\dot{\varepsilon}_{vol}$
is the volumetric strain rate in the porous medium,

$\dot{\varepsilon}_w^{th}$
is the volumetric thermal expansion rate in the porous fluid,

$\dot{\varepsilon}_g^{th}$
is the volumetric thermal expansion rate in the porous solid skeleton,

v_w
is the average velocity of the fluid relative to the solid phase,

α
is the Biot parameter defined as $\alpha = 1 - K/K_g$, and

K
is the bulk elastic modulus of the porous material skeleton.

Under undrained conditions, the fluid velocity terms can be set to zero in the continuity equation:

$$\frac{1}{Q}\frac{du_w}{dt} = -\alpha\left(\dot{\varepsilon}_{vol} - \dot{\varepsilon}_g^{th}\right) + n\left(\dot{\varepsilon}_w^{th} - \dot{\varepsilon}_g^{th}\right),$$

where

$$\frac{1}{Q} = \frac{n}{K_w} + \frac{\alpha - n}{K_g}.$$

Integrating in time, the incremental pore fluid pressure is given by

$$\Delta u_w = Q\left[-\alpha\left(\Delta\varepsilon_{vol} - \Delta\varepsilon_g^{th}\right) + n\left(\Delta\varepsilon_w^{th} - \Delta\varepsilon_g^{th}\right)\right].$$

The porosity evolution is

$$1 - n = \left(1 - n^0\right)\frac{J_g}{J},$$

where the superscript 0 indicates values in the reference configuration. J is the ratio of the medium's volume in the current configuration to its volume in the reference configuration, and J_g is the ratio of the current to reference volume for the grains.

The total stress and strain are given by

$$\boldsymbol{\sigma} = \overline{\boldsymbol{\sigma}} - u_w I$$

$$\varepsilon = \overline{\varepsilon} - \frac{1}{3}\left(\frac{u_w}{K_g} - \varepsilon_g^{th}\right)I$$

$$\overline{\boldsymbol{\sigma}} = \overline{\boldsymbol{\sigma}}\left(\text{strain history, temperature, state variables}\right).$$

In the above expression $\boldsymbol{\sigma}$ is the total stress, and $\overline{\boldsymbol{\sigma}}$ is the effective stress that is a function of the effective strain $\overline{\boldsymbol{\epsilon}}$ and the constitutive response of the porous material skeleton (see *Effective stress principle for porous media*).

Initial Conditions

You can apply initial conditions as defined in *Initial Conditions*.

Defining Initial Pore Fluid Pressures

You can define initial values of pore fluid pressures, u_w, at the nodes.

Input File Usage: **INITIAL CONDITIONS*, TYPE=PORE PRESSURE

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Pore pressure** for the **Types for Selected Step**

Defining Initial Void Ratios

You can give initial values of the void ratio, *e*, at the nodes. Abaqus/Explicit defines the void ratio as the ratio of the volume of voids to the volume of solid material (see *Effective stress principle for porous media*). The evolution of void ratio is governed by the deformation of the different phases of the material, as discussed in detail in *Constitutive behavior in a porous medium*.

Input File Usage: **INITIAL CONDITIONS*, TYPE=RATIO

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Void ratio** for the **Types for Selected Step**

Defining Initial Porosity

Alternatively, you can define initial values of the porosity, n, which can be specified at the element level. Abaqus/Explicit defines the porosity as the ratio of the volume of voids to the total volume (see *Effective stress principle for porous media*). The evolution of porosity is governed by the deformation of the different phases of the material, as discussed in detail in *Constitutive behavior in a porous medium*.

The conversion relationships between porosity and void ratio are given by

$$e = \frac{n}{1-n},$$
$$n = \frac{e}{1+e}.$$

When you define both the initial porosity and the initial void ratio at the same time, Abaqus/Explicit overwrites the value of the void ratio with the value of the porosity.

Input File Usage: **INITIAL CONDITIONS*, TYPE=POROSITY

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **porosity** for the **Types for Selected Step**

Boundary Conditions

In an undrained pore fluid flow and stress analysis, you can apply boundary conditions to the displacement degrees of freedom (1-6). You cannot apply boundary conditions to the pore pressure because pore pressure is not a degree of freedom.

Loads

You can use any of the loading types that can be prescribed during a pure stress analysis in an undrained pore fluid flow and stress analysis.

Predefined Fields

You can specify predefined temperature fields and field variables as described in *Predefined Fields* and *Explicit Dynamic Analysis*. You cannot specify nodal pore fluid pressures as predefined field variables in undrained analyses (see *Pore Fluid Pressure*).

Material Options

You can use any of the mechanical constitutive models available in Abaqus/Explicit in an undrained pore fluid flow and stress analysis.

You must include the densities of the solid material and the permeating fluid in the material definition separately (see *Density*).

You must define the compressibility of the solid grains and of the permeating fluid in the material definition (see *Porous Bulk Moduli*). Specifying very large values of the bulk moduli for the grain and fluid materials (for example, nearly incompressible behavior) will have a significant negative effect on the stable time increment.

You can define the thermal expansion for the solid grains and the permeating fluid separately (see *Thermal Expansion*).

If you did not define either the density of the permeating fluid or the porous bulk moduli, Abaqus/Explicit performs a pure stress analysis.

Input File Usage: To define the bulk moduli of the solid grains and the permeating fluid, use the following option:

**POROUS BULK MODULI*

To define the density and thermal expansion of the permeating fluid, use the following options:

**DENSITY*, PORE FLUID

**EXPANSION*, PORE FLUID

To define the density and thermal expansion of the solid material, use the following options:

**DENSITY*,

**EXPANSION*

Abaqus/CAE Usage: Defining the density of the permeating fluid is not supported in Abaqus/CAE.

To define density of the solid material, use the following option:

Property module: material editor: **General->Density**

To define the porous bulk moduli, use the following option:

Property module: material editor: **Other->Pore Fluid->Porous Bulk Moduli**

To define the thermal expansion of the permeating fluid:

Property module: material editor: **Other->Pore Fluid->Pore Fluid Expansion**

To define the thermal expansion of the solid material:

Property module: material editor: **Mechanical->Expansion**

Elements

You can use any solid (continuum) stress/displacement element that supports a three-dimensional stress state, except the incompatible mode element C3D8I, with a permeating fluid in undrained pore fluid flow and stress analysis. You can include C3D8I elements and other elements in the analysis as long as they do not contain a material with a permeating fluid.

Output

The element output available for an undrained pore fluid flow and stress analysis includes the usual mechanical quantities such as (effective) stress; strain; energies; and the values of state, field, and user-defined variables. In addition, the following quantities associated with pore fluid pressure are available:

POR
Pore pressure, u_w.

PORVAVG
Volume fraction weighted average pore pressure, u_w for Eulerian elements.

For more information, see *Abaqus/Explicit Output Variable Identifiers*.

Input File Template

```
*HEADING
 …
*MATERIAL, NAME=name
*ELASTIC
 …
*DENSITY
Data lines to define density
*DENSITY, PORE FLUID
Data lines to define density of fluid
 *POROUS BULK MODULI
Data line to define porous bulk moduli
 …
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=PORE PRESSURE
Data lines to specify initial pore pressure
*INITIAL CONDITIONS, TYPE=POROSITY
Data lines to specify initial porosity
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**********
*STEP
*DYNAMIC, EXPLICIT, PORE PRESSURE=UNDRAINED
Data line to specify the time period of the step
*BOUNDARY, AMPLITUDE=name
Data lines to describe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loading
Data lines to specify predefined fields
*FILE OUTPUT, NUMBER INTERVAL=n
*EL FILE
Data line specifying element output variables
*NODE FILE
Data line specifying node output variables
*ENERGY FILE
*OUTPUT, FIELD, NUMBER INTERVAL=n
*ELEMENT OUTPUT
Data line specifying element output variables
*NODE OUTPUT
Data line specifying node output variables
*OUTPUT, HISTORY, TIME INTERVAL=t
*ELEMENT OUTPUT, ELSET=element set name
Data line specifying element output variables
*NODE OUTPUT, NSET=node set name
Data line specifying node output variables
*ENERGY OUTPUT
```

Data line specifying energy output variables
```
*END STEP
```

References

- Hamann, T., and J. Grabe, "A Simple Dynamic Approach for the Numerical Modelling of Soil as a Two-Phase Material," *Geotechnik*, vol. 36, pp. 180–191, 2013.

Geostatic Stress State

Products: Abaqus/Standard Abaqus/CAE

Overview

A geostatic stress field procedure:

- is used to verify that the initial geostatic stress field is in equilibrium with applied loads and boundary conditions and to iterate, if necessary, to obtain equilibrium;
- accounts for pore pressure degrees of freedom when pore pressure elements are used, and accounts for temperature degrees of freedom when coupled temperature–pore pressure elements are used;
- is usually the first step of a geotechnical analysis, followed by a coupled pore fluid diffusion/stress (with or without heat transfer) or static analysis procedure; and
- can be linear or nonlinear.

References:

- *Defining an Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*
- **GEOSTATIC*
- *Configuring a geostatic stress field procedure*

Establishing Geostatic Equilibrium

The geostatic procedure is normally used as the first step of a geotechnical analysis; in such cases gravity loads are applied during this step. Ideally, the loads and initial stresses should exactly equilibrate and produce zero deformations. However, in complex problems it may be difficult to specify initial stresses and loads that equilibrate exactly.

Abaqus/Standard provides two procedures for establishing the initial equilibrium. The first procedure is applicable to problems for which the initial stress state is known at least approximately. The second, enhanced, procedure is also applicable for cases in which the initial stresses are not known; it is supported for only a limited number of elements and materials.

Establishing Equilibrium When the Initial Stress State Is Approximately Known

The geostatic procedure requires that the initial stresses are close to the equilibrium state; otherwise, the displacements corresponding to the equilibrium state might be large. Abaqus/Standard checks for equilibrium during the geostatic procedure and iterates, if needed, to obtain a stress state that equilibrates the prescribed boundary conditions and loads. This stress state, which is a modification of the stress field defined by the initial conditions (*Initial Conditions*), is then used as the initial stress field in a subsequent static or coupled pore fluid diffusion/stress (with or without heat transfer) analysis.

If the stresses given as initial conditions are far from equilibrium under the geostatic loading and there is some nonlinearity in the problem definition, this iteration process may fail. Therefore, you should ensure that the initial stresses are reasonably close to equilibrium.

If the deformations produced during the geostatic step are significant compared to the deformations caused by subsequent loading, the definition of the initial state should be reexamined.

If heat transfer is modeled during the geostatic step through the use of coupled temperature–pore pressure elements, the initial temperature field and thermal loads, if specified, must be such that the system is relatively close to a state of thermal equilibrium. Steady-state heat transfer is assumed during a geostatic step.

Input File Usage: **GEOSTATIC*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Geostatic**

Establishing Equilibrium When the Initial Stress State Is Unknown

To obtain equilibrium in cases when the initial stress state is unknown or is known only approximately, you can invoke an enhanced procedure. Abaqus automatically computes the equilibrium corresponding to the initial loads and the initial configuration, allowing only small displacements within user-specified tolerances. (The default tolerance is 10^{-5}.) The procedure is available with a limited number of elements and materials and is intended to be used in analyses in which the material response is primarily elastic; that is, inelastic deformations are small.

The procedure is supported for both geometrically linear and geometrically nonlinear analyses. However, in general, the performance in the geometrically linear case will be better. Therefore, it might be advantageous to obtain the initial equilibrium in a geometrically linear step, even though a geometrically nonlinear analysis is performed in subsequent steps.

Input File Usage: Use the following option to invoke the enhanced procedure:

**GEOSTATIC*, UTOL=*displacement tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Geostatic**: **Incrementation** tabbed page: **Automatic**: **Max. displacement change**

Limitations

The following limitations apply to the enhanced procedure:

- It is supported only for a limited number of elements (see *Elements* below) and materials (see *Material Options* below). When the procedure is used with nonsupported elements or material models, Abaqus issues a warning message. In this case it is the user's responsibility to ensure that the displacement tolerances are larger than the displacements in the analysis; otherwise, convergence problems may occur.
- It can be used in a restart analysis only if it had been used in the previous analysis.
- If the enhanced procedure is used with elements that have pore pressure degrees of freedom, the results might depend on the values of initial stresses specified.

Optional Modeling of Coupled Heat Transfer

When coupled temperature–pore pressure elements are used, heat transfer is modeled in these elements by default. However, you may optionally choose to switch off heat transfer within these elements during a geostatic step. This feature may be helpful in reducing computation time if temperature and associated heat flow effects are not important.

Input File Usage: Use the following option to suppress heat transfer modeling:

**GEOSTATIC*, HEAT=NO

Abaqus/CAE Usage: Switching off the heat transfer part of the physics is not supported in Abaqus/CAE.

Vertical Equilibrium in a Porous Medium

Most geotechnical problems begin from a geostatic state, which is a steady-state equilibrium configuration of the undisturbed soil or rock body under geostatic loading. The equilibrium state usually includes both horizontal and vertical stress components. It is important to establish these initial conditions correctly so that the problem begins from an equilibrium state. Since such problems often involve fully or partially saturated flow, the initial void ratio of the porous medium, e^0, the initial pore pressure, u_w, and the initial effective stress must all be defined.

If the magnitude and direction of the gravitational loading are defined by using the gravity distributed load type, a total, rather than excess, pore pressure solution is used (see *Coupled Pore Fluid Diffusion and Stress Analysis*). This discussion is based on the total pore pressure formulation.

The z-axis points vertically in this discussion, and atmospheric pressure is neglected. We assume that the pore fluid is in hydrostatic equilibrium during the geostatic procedure so that

$$\frac{du_w}{dz} = -\gamma_w,$$

where γ_w is the user-defined specific weight of the pore fluid (see *Permeability*). (The pore fluid is not in hydrostatic equilibrium if there is significant steady-state flow of pore fluid through the porous medium: in that case a steady-state coupled pore fluid diffusion/stress analysis must be performed to establish the initial conditions for any subsequent transient calculations—see *Coupled Pore Fluid Diffusion and Stress Analysis*.) If we also take γ_w to be independent of z (which is usually the case, since the fluid is almost incompressible), this equation can be integrated to define

$$u_w = \gamma_w \left(z_w^0 - z\right),$$

where z_w^0 is the height of the *phreatic surface*, at which $u_w = 0$ and above which $u_w < 0$ and the pore fluid is only partially saturated.

We usually assume that there are no significant shear stresses τ_{xz}, τ_{yz}. Then, equilibrium in the vertical direction is

$$\frac{d\sigma_{zz}}{dz} = \rho g + s n^0 \gamma_w,$$

where ρ is the dry density of the porous solid material (the dry mass per unit volume), g is the gravitational acceleration, n^0 is the initial porosity of the material, and s is the saturation, $0 \le s \le 1.0$ (see *Permeability*). Since porosity is the ratio of pore volume to total volume and the void ratio is the ratio of pore volume to solids volume, n^0 is defined from the initial void ratio by

$$n^0 = \frac{e^0}{1+e^0}.$$

Abaqus/Standard requires that the initial value of the effective stress, $\overline{\boldsymbol{\sigma}}$, be given as an initial condition (*Initial Conditions*). Effective stress is defined from the total stress, $\boldsymbol{\sigma}$, by

$$\overline{\boldsymbol{\sigma}} = \boldsymbol{\sigma} + s u_w \mathbf{I},$$

where $\mathbf{I}$ is a unit matrix. Combining this definition with the equilibrium statement in the z-direction and hydrostatic equilibrium in the pore fluid gives

$$\frac{d\bar{\sigma}_{zz}}{dz} = \rho g - \gamma_w \left(s\left(1-n^0\right) - \frac{ds}{dz}\left(z_w^0 - z\right)\right) \quad \text{for} \quad z < z_1^0, \quad \text{and}$$

$$\frac{d\overline{\sigma}_{zz}}{dz} = \rho g \quad \text{for} \quad z_1^0 \le z,$$

again using the assumption that γ_w is independent of z. z_1^0 is the position of the surface that separates the dry soil from the partially saturated soil. The soil is assumed to be dry ($s = 0$) for $z_1^0 < z$, and it is assumed to be partially saturated for $z_w^0 < z < z_1^0$ and fully saturated for $z \le z_w^0$.

In many cases s is constant. For example, in fully saturated flow $s = 1.0$ everywhere below the phreatic surface. If we further assume that the initial porosity, n^0, and the dry density of the porous medium, ρ, are also constant, the above equation is readily integrated to give

$$\overline{\sigma}_{zz} = \rho g \left(z - z^0\right) - \gamma_w s \left(1 - n^0\right) \left(z - z_w^0\right) \quad \text{for} \quad z < z_1^0 , \quad \text{and}$$
$$\overline{\sigma}_{zz} = \rho g \left(z - z^0\right) \quad \text{for} \quad z_1^0 \le z,$$

where z^0 is the position of the surface of the porous medium, $z_w^0 < z^0$.

In more complicated cases where s, n^0, and/or ρ vary with height, the equation must be integrated in the vertical direction to define the initial values of $\overline{\sigma}_{zz}(z)$.

Horizontal Equilibrium in a Porous Medium

In many geotechnical applications there is also horizontal stress, typically caused by tectonic action. If the pore fluid is under hydrostatic equilibrium and $\tau_{xz} = \tau_{yz} = 0$, equilibrium in the horizontal directions requires that the horizontal components of effective stress do not vary with horizontal position: $\overline{\sigma}_h(z)$ only, where $\overline{\sigma}_h$ is any horizontal component of effective stress.

Soils Mechanics Effective Stress Versus Rock Mechanics Effective Stress

There are two different conventions to define the effective stress. The effective stress, $\overline{\boldsymbol{\sigma}}$, defined above, is often referred to as the soils mechanics effective stress. Another form of effective stress that takes into account the effect of the bulk modulus of the solid grains is often referred to as the rock mechanics effective stress, $\overline{\boldsymbol{\sigma}}^{**}$. The rock mechanics effective stress is used to evaluate the damage state of the material if a material damage model is present or the element is enriched. The material plasticity constitutive behavior is always computed based on the soils mechanics effective stress regardless of the material damage state. The rock mechanics effective stress, $\overline{\boldsymbol{\sigma}}^{**}$, is related to the soils mechanics effective stress, $\overline{\boldsymbol{\sigma}}$, by

$$\overline{\boldsymbol{\sigma}}^{**} = \overline{\boldsymbol{\sigma}} - (1 - \alpha) s u_w \mathbf{I},$$

where α is the so-called Biot's coefficient. Biot's coefficient is defined as $\alpha = 1 - \frac{K}{K_s}$, where K is the bulk elastic modulus of the porous material skeleton and K_s is the bulk elastic modulus of the solid grains.

Initial Conditions

The initial effective geostatic stress field, $\overline{\boldsymbol{\sigma}}$, is given by defining initial stress conditions. This soils mechanics effective stress is then converted into the rock mechanics effective stress as defined above to evaluate the damage state of the material if a material damage model is present or the element is enriched. Unless the enhanced procedure is used, the initial state of stress must be close to being in equilibrium with the applied loads and boundary conditions. See *Initial Conditions*.

You can specify that the initial stresses vary only with elevation, as described in *Initial Conditions*. In this case the horizontal stress is typically assumed to be a fraction of the vertical stress: those fractions are defined in the *x*- and *y*-directions.

In problems involving partially or fully saturated porous media, initial pore fluid pressures, u_w, void ratios, e^0, and saturation values, *s*, must be given (see *Coupled Pore Fluid Diffusion and Stress Analysis*).

In partially saturated cases the initial pore pressure and saturation values must lie on or between the absorption and exsorption curves (see *Sorption*). A partially saturated problem is illustrated in *Wicking in a partially saturated porous medium*.

The initial effective stress/pore pressure conditions defined for an element are assumed to be acting on the initial configuration of the element. If the initial effective stress/pore pressure conditions are removed during the step, the element returns to a stress-free configuration that is different from the initial one. Since displacements and total strain output are measured relative to the initial configuration, the stress-free configuration will have nonzero values for the displacement and total strain fields that will depend on the initial conditions. While it is easy to verify the above behavior analytically in a one-element problem subjected to an initial stress and pore pressure field, the situation in a complex boundary value problem is determined by other factors that may make it difficult to resolve analytically.

You can also specify initial temperatures in the model if heat transfer is modeled during the geostatic procedure.

Boundary Conditions

Boundary conditions can be applied to displacement degrees of freedom 1–6 and to pore pressure degree of freedom 8 (*Boundary Conditions*). If coupled temperature–pore pressure elements are used, boundary conditions on temperature degree of freedom 11 can also be applied to nodes belonging to these elements. If the enhanced procedure is used and nonzero boundary conditions are applied, it is the user's responsibility to ensure that the displacements corresponding to the tolerances specified are larger than the displacements in the analysis; otherwise, the displacements at the nonzero boundary nodes will be reset to zero with the tolerances specified.

The boundary conditions should be in equilibrium with the initial stresses and applied loads. If the horizontal stress is nonzero, horizontal equilibrium must be maintained by fixing the boundary conditions on any nonhorizontal edges of the finite element model in the horizontal direction or by using infinite elements (*Infinite Elements*). If heat transfer is modeled, the temperature boundary conditions should be in equilibrium with the initial temperature field and applied thermal loads.

Loads

The following loading types can be prescribed in a geostatic stress field procedure:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can also be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*. The magnitude and direction of gravitational loading are defined by using the gravity or body force distributed load types.
- Pore fluid flow is controlled as described in *Pore Fluid Flow*.

If heat transfer is modeled, the following types of thermal loading can also be prescribed (*Thermal Loads*). These loads are not supported in Abaqus/CAE during a geostatic analysis.

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Predefined Fields

The following predefined fields can be specified in a geostatic stress field procedure, as described in *Predefined Fields*:

- For a geostatic analysis that does not model heat transfer and uses regular pore pressure elements, temperature is not a degree of freedom and nodal temperatures can be specified.
- Predefined temperature fields are not allowed in a geostatic analysis that also models heat transfer. Boundary conditions should be used instead to specify temperatures, as described earlier.
- The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any.

Material Options

Any of the mechanical constitutive models available in Abaqus/Standard can be used to model the porous solid material. However, the enhanced procedure can be used only with the elastic, porous elastic, extended Cam-clay plasticity, and Mohr-Coulomb plasticity models. Use of a nonsupported material model with this procedure may lead to poor convergence or no convergence if displacements are larger than the displacements corresponding to the tolerances specified. Abaqus will issue a warning message if the procedure is used with a nonsupported material model.

If a porous medium will be analyzed subsequent to the geostatic procedure, pore fluid flow quantities such as permeability and sorption should be defined (see *About Pore Fluid Flow Properties*).

If heat transfer is modeled, thermal properties such as conductivity, specific heat, and density should be defined for both the solid and the pore fluid phases (see *Thermal Properties if Heat Transfer is Modeled* for details on how to specify separate thermal properties for the two phases).

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a geostatic procedure. Continuum pore pressure elements can also be used for modeling fluid in a deforming porous medium. These elements have pore pressure degree of freedom 8 in addition to displacement degrees of freedom 1–3. However, the enhanced procedure can be used only with continuum and cohesive elements with pore pressure degrees of freedom and the corresponding stress/displacements elements. Use of nonsupported elements with this procedure may lead to poor convergence or no convergence if displacements are larger than the displacements corresponding to the tolerances specified. Abaqus will issue a warning message if the procedure is used with a nonsupported element.

Continuum elements that couple temperature, pore pressure, and displacement can be used if heat transfer needs to be modeled. These elements have temperature degree of freedom 11 in addition to pore pressure degree of freedom 8 and displacement degrees of freedom 1–3. See *Choosing the Appropriate Element for an Analysis Type* for more information.

Output

The element output available for a coupled pore fluid diffusion/stress analysis includes the usual mechanical quantities such as (effective) stress; strain; energies; and the values of state, field, and user-defined variables. In addition, the following quantities associated with pore fluid flow are available:

VOIDR Void ratio, e.

POR Pore pressure, u_w.

SAT Saturation, s.

GELVR Gel volume ratio, n_t.

FLUVR Total fluid volume ratio, n_f.

FLVEL Magnitude and components of the pore fluid effective velocity vector, $\mathbf{f}$.

FLVELM Magnitude, $\mathbf{f}$, of the pore fluid effective velocity vector.

FLVEL*n* Component n of the pore fluid effective velocity vector (n=1, 2, 3).

If heat transfer is modeled, the following element output variables associated with heat transfer are also available:

HFL Magnitude and components of the heat flux vector.

HFL*n* Component n of the heat flux vector (n=1, 2, 3).

HFLM Magnitude of the heat flux vector.

TEMP Integration point temperatures.

TEMPR Integration point temperature rate.

GRADT Temperature gradient vector.

GRADT*n* Component n of the temperature gradient (n=1,2,3).

The nodal output available includes the usual mechanical quantities such as displacements, reaction forces, and coordinates. In addition, the following quantities associated with pore fluid flow are available:

POR Pore pressure at a node.

RVF Reaction fluid volume flux due to prescribed pressure. This flux is the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pressure boundary condition. A positive value of RVF indicates fluid is entering the model.

If heat transfer is modeled, the following nodal output variables associated with heat transfer are also available:

NT Nodal point temperatures.

RFL Reaction flux values due to prescribed temperature.

RFL*n* Reaction flux value n at a node (n=11, 12, …).

CFL Concentrated flux values.

CFL*n*	Concentrated flux value *n* at a node (*n*=11, 12, …).
SROCK*ij*	Rock mechanics effective stress tensor.

All of the output variable identifiers are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
…
*MATERIAL, NAME=mat1
Data lines to define mechanical properties of the solid material
…
*DENSITY
Data lines to define the density of the dry material
*PERMEABILITY, SPECIFIC=γw
Data lines to define permeability, k, as a function of the void ratio, e
*CONDUCTIVITY
Data lines to define thermal conductivity of the solid grains if heat transfer is modeled
*CONDUCTIVITY,TYPE=ISO, PORE FLUID
Data lines to define thermal conductivity of the permeating fluid if heat transfer is modeled
*SPECIFIC HEAT
Data lines to define specific heat of the solid grains if transient heat transfer is modeled in a
subsequent step
*SPECIFIC HEAT,PORE FLUID
Data lines to define specific heat of the permeating fluid if transient heat transfer is modeled in a subsequent
step
*DENSITY
Data lines to define density of the solid grains if transient heat transfer is modeled in a subsequent
step
*DENSITY,PORE FLUID
Data lines to define density of the permeating fluid if transient heat transfer is modeled in a
subsequent step
*LATENT HEAT
Data lines to define latent heat of the solid grains if phase change due to temperature change is modeled
*LATENT HEAT,PORE FLUID
Data lines to define latent heat of the permeating fluid if phase change due to temperature change
is modeled
…
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC
Data lines to define the initial stress state
*INITIAL CONDITIONS, TYPE=PORE PRESSURE
Data lines to define initial values of pore fluid pressures
*INITIAL CONDITIONS, TYPE=RATIO
Data lines to define initial values of the void ratio
*INITIAL CONDITIONS, TYPE=SATURATION
Data lines to define initial saturation
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data lines to define initial temperature
*BOUNDARY
Data lines to define zero-valued boundary conditions
**
*STEP
*GEOSTATIC
*CLOAD and/or *DLOAD and/or *DSLOAD
```

Data lines to specify mechanical loading
**FLOW* and/or **SFLOW* and/or **DFLOW* and/or **DSFLOW*
Data lines to specify pore fluid flow
**CFLUX* and/or **DFLUX*
Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled
**BOUNDARY*
Data lines to specify displacements or pore pressures
**END STEP*

Mass Diffusion Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A mass diffusion analysis:

- models the transient or steady-state diffusion of one material through another, such as the diffusion of hydrogen through a metal;
- requires the use of mass diffusion elements; and
- can be used to model temperature and/or pressure-driven mass diffusion.

References:

- *Defining an Analysis*
- **MASS DIFFUSION*
- *Configuring a mass diffusion procedure*
- *Creating and modifying prescribed conditions*
- *Defining a concentrated concentration flux*
- *Defining a body concentration flux*
- *Defining a surface concentration flux*

Governing Equations

The governing equations for mass diffusion are an extension of Fick's equations: they allow for nonuniform solubility of the diffusing substance in the base material and for mass diffusion driven by gradients of temperature and pressure. The basic solution variable (used as the degree of freedom at the nodes of the mesh) is the

"normalized concentration" (often also referred to as the "activity" of the diffusing material), $\phi \overset{\text{def}}{=} c/s$, where c is the mass concentration of the diffusing material and s is its solubility in the base material. Therefore, when the mesh includes dissimilar materials that share nodes, the normalized concentration is continuous across the interface between the different materials.

For example, a diatomic gas that dissociates during diffusion can be described using Sievert's law: $c = s(p/p_0)^{\frac{1}{2}}$, where p is the partial pressure of the diffusing gas, and the value of p_0 is customarily one atmosphere (for historical consistency). Combining Sievert's law with the definition of normalized concentration given earlier, $\phi = c/s = (p/p_0)^{\frac{1}{2}}$. Equilibrium requires the partial pressure to be continuous across an interface, so normalized concentration will be continuous as well. If an expression other than Sievert's law defines the relationship between concentration and partial pressure for a diffusing material, solubility should be defined accordingly.

The diffusion problem is defined from the requirement of mass conservation for the diffusing phase:

$$\int_V \frac{dc}{dt}\, dV + \int_S \mathbf{n} \cdot \mathbf{J}\, dS = 0,$$

where V is any volume whose surface is S, $\mathbf{n}$ is the outward normal to S, $\mathbf{J}$ is the flux of concentration of the diffusing phase, and $\mathbf{n} \cdot \mathbf{J}$ is the concentration flux leaving S.

Diffusion is assumed to be driven by the gradient of a general chemical potential, which gives the behavior

$$\mathbf{J} = -s\mathbf{D} \cdot \left[\frac{\partial \phi}{\partial \mathbf{x}} + \kappa_s \frac{\partial}{\partial \mathbf{x}} \left(\ln\left(\theta - \theta^Z\right) \right) + \kappa_p \frac{\partial p}{\partial \mathbf{x}} \right],$$

where $\mathbf{D}\left(c, \theta, \mathbf{f}\right)$ is the diffusivity; $s\left(\theta, \mathbf{f}\right)$ is the solubility; $\kappa_s\left(c, \theta, \mathbf{f}\right)$ is the "Soret effect" factor, providing diffusion because of temperature gradient; θ is the temperature; θ^Z is the value of absolute zero on the temperature scale being used; $\kappa_p\left(c, \theta, \mathbf{f}\right)$ is the pressure stress factor, providing diffusion driven by the gradient of the equivalent pressure stress, $p \stackrel{\text{def}}{=} -\text{trace}\left(\boldsymbol{\sigma}\right)/3$; $\boldsymbol{\sigma}$ is stress; and $\mathbf{f}$ are any predefined field variables.

Whenever D, κ_s, or κ_p depends on concentration, the problem becomes nonlinear and the system of equations becomes nonsymmetric. In practical cases the dependence on concentration is quite strong, so the nonsymmetric matrix storage and solution scheme is invoked automatically when a mass diffusion analysis is performed (see *Defining an Analysis*).

Fick's Law

Mass diffusion behavior is often described by Fick's law (Crank, 1956):

$$\mathbf{J} = -\mathbf{D} \cdot \frac{\partial c}{\partial \mathbf{x}}.$$

Fick's law is offered in Abaqus/Standard as a special case of the general chemical potential relation. To establish the relationship between Fick's law and the general chemical potential, we write Fick's law as

$$\mathbf{J} = -\mathbf{D} \cdot \left(s \frac{\partial \phi}{\partial \mathbf{x}} + \phi \frac{\partial s}{\partial \mathbf{x}} \right).$$

In most practical cases $s = s\left(\theta\right)$, and we can write

$$\mathbf{J} = -s\mathbf{D} \cdot \frac{\partial \phi}{\partial \mathbf{x}} - \mathbf{D} \cdot \frac{c}{s} \frac{\partial s}{\partial \theta} \frac{\partial \theta}{\partial \mathbf{x}}.$$

The two terms in this equation describe the normalized concentration and temperature-driven diffusion, respectively. The normalized concentration-driven diffusion term is identical to that given in the general relation. The temperature-driven diffusion term in Fick's law is recovered in the general relation if

$$\kappa_s = \frac{c\left(\theta - \theta^Z\right)}{s^2} \frac{\partial s}{\partial \theta} \quad .$$

This conversion is done automatically in Abaqus/Standard when you request Fick's law (see *Diffusivity*).

An extended form of Fick's law can also be chosen by specifying a nonzero value for κ_p:

$$\mathbf{J} = -\mathbf{D} \cdot \left(\frac{\partial c}{\partial \mathbf{x}} + s\kappa_p \frac{\partial p}{\partial \mathbf{x}} \right).$$

In this case Abaqus/Standard will still define κ_s automatically as discussed earlier.

Units

The units of concentration are commonly given as parts per million (P). Similarly, the units of solubility are P. The Soret effect factor is dimensionless. The units of the pressure stress factor are $F^{-1}L^2$, where F is force and L is length; and the units of equivalent pressure stress are FL^{-2}. The diffusivity, $\mathbf{D}$, has units of L^2T^{-1}, where T is time. The concentration flux, $\mathbf{J}$, then has units of PLT^{-1}; and the concentration volumetric flux, $Q = \int_S \mathbf{n} \cdot \mathbf{J}\, dS$, has units of PL^3T^{-1}.

Steady-State Analysis

Steady-state mass diffusion analysis provides the steady-state solution directly: the rate of change of concentration with respect to time is omitted from the governing diffusion equation in steady-state analysis. In nonlinear cases iteration may be necessary to achieve a converged solution.

Since the rate term is removed from the governing equations, the steady-state problem has no intrinsic physically meaningful time scale; nevertheless, you may assign a "time" scale to the analysis step. This time scale is often convenient for output identification and for specifying prescribed normalized concentrations and fluxes with varying magnitudes. Thus, when steady-state analysis is chosen, you specify a "time" increment and a "time" period for the step; Abaqus/Standard then increments through the step accordingly. If a steady-state analysis step is to be followed by a transient analysis step and total time is used in amplitude definitions (*Amplitude Curves*), the time period should be defined to be negligibly small in the steady-state step. For more details on time scales and time stepping, see *Defining an Analysis*.

Input File Usage: **MASS DIFFUSION*, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Steady state**

Transient Analysis

Time integration in transient diffusion analysis is done with the backward Euler method (also referred to as the modified Crank-Nicholson operator). This method is unconditionally stable for linear problems.

Automatic or fixed time incrementation can be used for transient analysis. The automatic time incrementation scheme is generally preferred because the response is usually simple diffusion: the rate of change of normalized concentration varies widely during the step and requires different time increments to maintain accuracy in the time integration.

Spurious Oscillations due to Small Time Increments

In transient mass diffusion analysis with second-order elements there is a relationship between the minimum usable time step and the element size. A simple guideline is

$$\Delta t > \frac{1}{6D}\Delta \ell^2,$$

where Δt is the time increment, D is the diffusivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used, spurious oscillations can appear in the solution. Abaqus/Standard provides no check on the initial time increment defined for a mass diffusion analysis; you must ensure that the given value does not violate the above criterion.

In transient analysis using first-order elements the solubility terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the normalized concentration changes occur.

Generally there is no upper limit on the time increment because the integration procedure is unconditionally stable unless nonlinearities cause numerical problems.

Automatic Incrementation

The automatic time incrementation scheme for mass diffusion problems is based on the user-specified maximum normalized concentration change allowed at any node during an increment, ΔC_{max}.

Input File Usage: **MASS DIFFUSION*, DCMAX=ΔC_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable normalized concentration change:** ΔC_{max}

Fixed Time Incrementation

If you choose fixed time incrementation, fixed time increments equal to the size of the user-specified initial time increment, Δt_0, will be used.

Input File Usage: **MASS DIFFUSION*
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**, **Increment size:** Δt_0

Ending a Transient Analysis

Transient mass diffusion analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or the time period ends, whichever comes first. Steady state is defined as the point in time when all normalized concentrations change at less than a user-defined rate.

Input File Usage: Use the following option to end the analysis when the time period is reached:

**MASS DIFFUSION*, END=PERIOD (default)

Use the following option to end the analysis based on the concentration change rate:

**MASS DIFFUSION*, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **End step when normalized concentration change rate is less than**

Initial Conditions

An initial normalized concentration of the diffusing material at specific nodes that belong to mass diffusion elements can be defined (*Initial Conditions*). For an analysis in which mass diffusion is driven by gradients of temperature and/or pressure (*Diffusivity*), the initial temperature and pressure stress fields in a model can also be defined.

Input File Usage: Use the following options:

**INITIAL CONDITIONS*, TYPE=CONCENTRATION for initial concentrations
**INITIAL CONDITIONS*, TYPE=TEMPERATURE for initial temperatures
**INITIAL CONDITIONS*, TYPE=PRESSURE STRESS for initial equivalent pressure stress

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Temperature** for the **Types for Selected Step**

Initial concentration and equivalent pressure stress are not supported in Abaqus/CAE.

Boundary Conditions

Boundary conditions can be applied to nodal degree of freedom 11 in any mass diffusion element to prescribe values of normalized concentration (*Boundary Conditions*). Such values can be specified as functions of time.

Any boundary condition changes to be applied during a mass diffusion step should be given in the respective step using appropriate amplitude definitions to specify their "time" variations (*Amplitude Curves*). If boundary conditions are specified for the step without amplitude references, they are assumed to change either linearly with "time" during the step or instantly at the start of the step, according to the user-specified or default time variation associated with the step (see *Defining an Analysis*).

Loads

Concentration fluxes are the only loads that can be applied in a mass diffusion analysis step.

Input File Usage: Use the following option to specify a concentrated concentration flux at a node:

**CFLUX*
node number or node set name, *degree of freedom*, *concentrated flux magnitude*

Use the following option to specify a distributed concentration flux acting on entire elements (body flux) or just on element faces (surface flux):

**DFLUX*
element number or element set name, BF or S*n*, *distributed flux magnitude*

Abaqus/CAE Usage: Use the following input to define a concentrated concentration flux at a node:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Concentrated concentration flux** for the **Types for Selected Step**: select region: **Magnitude**: *concentrated flux magnitude*

Use the following input to define a distributed concentration flux acting on entire elements (body flux) or just on element faces (surface flux):

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Body concentration flux** or **Surface concentration flux** for the **Types for Selected Step**: **Distribution**: **Uniform** or select an analytical field, **Magnitude**: *distributed flux magnitude*

Modifying or Removing Concentration Fluxes

Concentrated or distributed concentration fluxes can be added, modified, or removed as described in *About Loads*.

Specifying Time-Dependent Concentration Fluxes

The magnitude of a concentrated or a distributed concentration flux can be controlled by referring to an amplitude curve (see *Amplitude Curves*). If different magnitude variations are needed for different fluxes, the flux definitions can be repeated, with each referring to its own amplitude curve.

Defining Nonuniform Distributed Concentration Fluxes in a User Subroutine

To define nonuniform distributed concentration fluxes, the variation of the flux magnitude throughout a step can be defined in user subroutine *DFLUX*. If a reference flux magnitude is specified directly, it will be ignored. As a result, any amplitude reference in the flux definition is also ignored.

Input File Usage: Use the following option to define a nonuniform distributed concentration body flux:

**DFLUX*
element number or element set, BFNU

Use the following option to define a nonuniform distributed concentration surface flux:

**DFLUX*
element number or element set, S*n*NU

Abaqus/CAE Usage: Use the following input to define a nonuniform distributed concentration body flux:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Body concentration flux** for the **Types for Selected Step**: select region: **Distribution: User-defined**

Use the following input to define a nonuniform distributed concentration surface flux:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Surface concentration flux** for the **Types for Selected Step**: select region: **Distribution: User-defined**

Predefined Fields

Predefined temperatures, equivalent pressure stresses, and field variables can be specified in a mass diffusion analysis.

Prescribing Temperatures

Temperatures are applied to nodes in temperature-driven mass diffusion analyses by defining a temperature field; absolute zero on the temperature scale used is defined as described in *Specifying the Value of Absolute Zero*. Alternatively, the temperature field can be obtained from a previous heat transfer analysis. Time-dependent temperature variations are possible with either approach.

A simple interface is provided that uses the Abaqus/Standard results file from the heat transfer analysis to define the temperature field at different times in the mass diffusion analysis. Abaqus/Standard assumes that the nodes in the mass diffusion analysis have the same numbers as the nodes in the previous heat transfer analysis. Values in the results file are ignored at nodes that exist in the heat transfer analysis but not in the mass diffusion analysis, and the temperatures at nodes that did not exist in the heat transfer analysis will not be set by reading the results file.

For specific details on prescribing temperatures, see *Predefined Temperature*.

Prescribing Equivalent Pressure Stresses

Equivalent pressure stress values can be given at nodes by specifying them directly as a predefined field in the mass diffusion analysis or indirectly by reading the equivalent pressure stresses from the results file of a previous stress/displacement, fully coupled temperature-displacement, or fully coupled thermal-electrical-structural analysis. Regardless of the manner in which they are specified, pressures should be entered according to the Abaqus convention that equivalent pressure stresses are positive when they are compressive.

A simple interface is provided that uses the Abaqus/Standard results file from a mechanical analysis to define the equivalent pressure stresses at different times in the mass diffusion analysis. Abaqus/Standard assumes that the nodes in the mass diffusion analysis have the same numbers as the nodes in the previous mechanical analysis. Values in the results file are ignored at nodes that exist in the mechanical analysis but not in the mass diffusion analysis, and the pressures at nodes that did not exist in the mechanical analysis will not be set by reading the results file.

For specific details on prescribing equivalent pressure stresses, see *Predefined Pressure Stress*.

Specifying Predefined Field Variables

You can specify values of predefined field variables during a mass diffusion analysis. These values affect only field-variable-dependent material properties, if any. See *Predefined Field Variables*.

Material Options

Both diffusivity (*Diffusivity*) and solubility (*Solubility*) must be defined in a mass diffusion analysis. Optionally, a Soret effect factor and a pressure stress factor can be defined to introduce mass diffusion caused by temperature and pressure gradients, respectively. The use of Fick's law also introduces temperature-driven mass diffusion since a Soret effect factor is calculated automatically.

Elements

Mass diffusion analysis can be performed using only the two-dimensional, three-dimensional, and axisymmetric solid elements that are included in the Abaqus/Standard heat transfer/mass diffusion element library.

Output

In addition to the standard output identifiers available in Abaqus/Standard (*Abaqus/Standard Output Variable Identifiers*), the following variables have special meaning in mass diffusion analyses:

Element integration point variables:

CONC Mass concentration.

ISOL Amount of solute at the integration point, calculated as the product of the mass concentration and the integration point volume.

MFL Magnitude and components of the concentration flux vector (excluding the terms due to pressure and temperature gradients).

MFLM Magnitude of the concentration flux vector.

MFL*n* Component n of the concentration flux vector (n = 1, 2, 3).

TEMP Magnitude of the applied temperature field.

Whole element variables:

ESOL Amount of solute in the element, calculated as the sum of ISOL over all the element integration points.

NFLUX Fluxes at the nodes of the element caused by mass diffusion in the element.

FLUXS Distributed mass flux applied to an element.

Whole or partial model variables:

SOL Amount of solute in the model or specified element set, calculated as the sum of ESOL over all the elements in the model or set.

Nodal variables:

CFL All concentrated flux values.

CFL*n* Concentrated flux value n at a node (n = 11).

NNC All normalized concentration values at a node.

NNC*n* Normalized concentration degree of freedom n at a node (n = 11).

RFL All reaction flux values (conjugate to normalized concentration).

RFL*n* Reaction flux value n at a node (n = 11) (conjugate to normalized concentration).

Input File Template

The following template is representative of a three-step mass diffusion analysis. The first step establishes an initial steady-state concentration distribution of a diffusing material. In the second step equivalent pressure stresses are read from a fully coupled temperature-displacement analysis and the transient mass diffusion response is obtained for the case of mechanical loading of the body. In the final step a temperature field is read from a fully coupled temperature-displacement analysis and the transient mass diffusion response is calculated for the case of heating and cooling the body in which diffusion occurs. An example problem that follows this template is *Thermomechanical diffusion of hydrogen in a bending beam*.

```
*HEADING
…
*MATERIAL,NAME=mat1
*SOLUBILITY
Data lines to define solubility
*DIFFUSIVITY
Data lines to define diffusivity
*KAPPA,TYPE=TEMP
Data lines to define diffusion driven by temperature gradients
*KAPPA,TYPE=PRESS
Data lines to define diffusion driven by gradients of equivalent pressure stress
*INITIAL CONDITIONS,TYPE=TEMPERATURE
Data lines to define an initial temperature field
*INITIAL CONDITIONS,TYPE=CONCENTRATION
Data lines to define initial nodal values of normalized concentration
*INITIAL CONDITIONS,TYPE=PRESSURE STRESS
Data lines to define initial nodal values of equivalent pressure stress
*AMPLITUDE,NAME=name
Data lines to define amplitude variations
**
*STEP
Step 1 - steady-state solution
*MASS DIFFUSION,STEADY STATE
Data line to define incrementation
*BOUNDARY
Data lines to prescribe nodal values of normalized concentration
*EL FILE
Data lines to define element integration output to the results file
*NODE FILE
Data lines to define nodal output to the results file
*END STEP
**
*STEP
Step 2 - transient analysis driven by pressure stress gradients
*MASS DIFFUSION,DCMAX=dcmax,END=SS
Data line to define incrementation
*BOUNDARY
Data lines to prescribe nodal values of normalized concentration
*PRESSURE STRESS,FILE=name
*EL FILE
Data lines to define element integration output to the results file
*NODE FILE
Data lines to define nodal output to the results file
*END STEP
**
*STEP
Step 3 - transient analysis driven by temperature gradients
```

```
*MASS DIFFUSION,DCMAX=dcmax,END=SS
Data line to define incrementation
*BOUNDARY
Data lines to prescribe nodal values of normalized concentration
*TEMPERATURE,FILE=name
*EL FILE
Data lines to define element integration output to the results file
*NODE FILE
Data lines to define nodal output to the results file
*END STEP
```

References

- Crank, J., The Mathematics of Diffusion, Clarendon Press, Oxford , 1956.

Coupled Acoustic-Structural Analysis

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

Overview

An acoustic analysis:

- is used to model sound propagation, emission, and radiation problems;
- can include incident wave loading to model effects such as underwater explosion (UNDEX) on structures interacting with fluids, airborne blast loading on structures, or sound waves impinging on a structure;
- in Abaqus/Explicit can include fluid undergoing cavitation when the absolute pressure drops to a limit value;
- is performed using one of the dynamic analysis procedures (*About Dynamic Analysis Procedures*);
- can be used to model an acoustic medium alone, as in the study of the natural frequencies of vibration of a cavity containing an acoustic fluid;
- can be used to model a coupled acoustic-structural system, as in the study of the noise level in a vehicle;
- can be used to model the sound transmitted through a coupled acoustic-structural system;
- requires the use of acoustic elements and, for coupled acoustic-structural analysis, a surface-based interaction using a tie constraint or, in Abaqus/Standard, acoustic interface elements;
- can be used to obtain the scattered wave solution directly under incident wave loading when the mechanical behavior of the fluid is linear;
- can be used to obtain a total wave solution (sum of the incident and the scattered waves) by selecting the total wave formulation, particularly when nonlinear fluid behavior such as cavitation is present in the acoustic medium;
- can be used to model problems where the acoustic medium interacts with a structure subjected to large static deformation;
- in Abaqus/Standard can be used with symmetric model generation (*Symmetric Model Generation*) and symmetric results transfer (*Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*);
- in Abaqus/Standard can be used with steady-state transport (*Steady-State Transport Analysis*) and an acoustic flow velocity (**ACOUSTIC FLOW VELOCITY*) to model acoustic perturbations of a moving fluid;
- in Abaqus/Standard can include a coupled structural-acoustic substructure that was previously defined (*Generating Substructures*);
- can be used to model both interior problems, where a structure surrounds one or more acoustic cavities, and exterior problems, where a structure is located in a fluid medium extending to infinity; and
- is applicable to any vibration or dynamic problem in a medium where the effects of shear stress are negligible.

A shock analysis:

- is used to model blast effects on structures;
- often requires double precision to avoid roundoff error when Abaqus/Explicit is used;
- may include acoustic elements to model the effects of fluid inertia and compressibility;
- may include virtual mass effects to model the effect of an incompressible fluid interacting with a pipe structure;

- is performed using one of the dynamic analysis procedures (*About Dynamic Analysis Procedures*);
- can be used to model both interior problems, where a structure surrounds one or more fluid cavities, and exterior problems, where a structure is located in a fluid medium extending to infinity; and
- in Abaqus/Explicit can include air blast loading on structures using the CONWEP model.

References:

- *Acoustic Medium*
- *Acoustic and Shock Loads*
- *Initial Conditions*
- *About ALE Adaptive Meshing*
- *Steady-State Transport Analysis*
- **ACOUSTIC FLOW VELOCITY*
- **ACOUSTIC WAVE FORMULATION*
- **ADAPTIVE MESH*
- **BEAM FLUID INERTIA*
- **CONWEP CHARGE PROPERTY*
- **IMPEDANCE*
- **IMPEDANCE PROPERTY*
- **INCIDENT WAVE*
- **INCIDENT WAVE INTERACTION*
- **INITIAL CONDITIONS*
- **SIMPEDANCE*
- **TIE*
- *Defining an acoustic pressure boundary condition*
- *Creating the submodel boundary condition*

Procedures Available for Acoustic Analysis

Acoustic elements model the propagation of acoustic waves and are active only in dynamic analysis procedures. They are used in the following procedures:

- Direct solution, steady-state, harmonic analysis. See *Direct-Solution Steady-State Dynamic Analysis*.
- Frequency analysis. See *Natural Frequency Extraction*.
- Subspace-based steady-state dynamic analysis. See *Subspace-Based Steady-State Dynamic Analysis*.
- Explicit dynamic analysis. See *Explicit Dynamic Analysis*.
- Direct time integration analysis. See *Implicit Dynamic Analysis Using Direct Integration*.
- Complex frequency analysis. See *Complex Eigenvalue Extraction*.
- Mode-based transient dynamic analysis. See *Transient Modal Dynamic Analysis*.
- Mode-based steady-state dynamic analysis. See *Mode-Based Steady-State Dynamic Analysis*.
- Dynamic fully coupled temperature-displacement analysis. See *Fully Coupled Thermal-Stress Analysis*.

In general, analysis with acoustic elements should be thought of as small-displacement linear perturbation analysis, in which the strain in the acoustic elements is strictly (or overwhelmingly) volumetric and small. In many applications the base state for the linear perturbation is simply ignored: for solid structures interacting with air or water, the initial stress (if any) in the air or water has negligible physical effect on the acoustic waves. Most engineering acoustic analyses, transient or steady state, are of this type.

An important exception is when the acoustic perturbation occurs in a gas or liquid with high-speed underlying flow. If the magnitude of the flow velocity is significant compared to the speed of sound in the fluid (i.e., the Mach number is much greater than zero), the propagation of waves is facilitated in the direction of flow and impeded in the direction against the flow. This phenomenon is the source of the well-known "Doppler effect." In Abaqus/Standard underlying flow effects are prescribed for nodes making up acoustic elements by specifying an acoustic flow velocity.

Acoustic elements can be used in a static analysis, but all acoustic effects will be ignored. A typical example is the air cavity in a tire/wheel assembly. In such a simulation the tire is subjected to inflation, rim mounting, and footprint loads prior to the coupled acoustic-structural analysis in which the acoustic response of the air cavity is determined. See *Defining ALE Adaptive Mesh Domains in Abaqus/Standard* and *ALE Adaptive Meshing and Remapping in Abaqus/Standard* for more information.

Acoustic elements also can be used in a substructure generation procedure to generate coupled structural-acoustic substructures. Only structural degrees of freedom can be retained. The retained eigenmodes must be selected when an acoustic-structural substructure is generated. In a static analysis involving a substructure containing acoustic elements, the results will differ from the results obtained in an equivalent static analysis without substructures. The reason is that the acoustic-structural coupling is taken into account in the substructure (leading to hydrostatic contributions of the acoustic fluid), while the coupling is ignored in a static analysis without substructures. More details on coupled structural-acoustic substructures can be found in *Generating Substructures*.

A volumetric drag coefficient, γ, can be defined to simulate fluid velocity-dependent pressure amplitude losses. These occur, for example, when the acoustic medium flows through a porous matrix that causes some resistance (see *Acoustic Medium*), such as a sound-deadening material like fiberglass insulation. For direct time integration

dynamic analysis we assume there are no significant spatial discontinuities in the quantity γ/ρ_f, where ρ_f is the density of the fluid (acoustic medium), and that the volumetric drag is small at acoustic-structural boundaries. These assumptions, which can limit the applicability of the analysis, are discussed further in *Coupled acoustic-structural medium analysis*.

The direct-solution steady-state dynamic harmonic response procedure is advantageous for acoustic-structural

sound propagation problems, because the gradient of γ/ρ_f need not be small and because acoustic-structural coupling and damping are not restricted in this formulation. If there is no damping or if damping can be neglected, factoring a real-only matrix can reduce computational time significantly; see *Direct-Solution Steady-State Dynamic Analysis* for details.

Some fluid-solid interaction analyses involve long-duration dynamic effects that more closely resemble structural dynamic analysis than wave propagation; that is, the important dynamics of the structure occur at a time scale that is long compared to the compressional wave speed of the solid medium and the acoustic wave speed of the fluid. Equivalently, in such cases, disturbances of interest in the structure propagate very slowly in comparison to waves in the fluid and compressional waves in the structure. In such instances, modeling of the structure using beams is common. When these structural elements interact with a surrounding fluid, the important fluid effect is due to motions associated with incompressible flow (see *Loading due to an incident dilatational wave field*). These motions result in a perceived inertia added to the structural beam; therefore, this case is usually referred to as the "virtual mass approximation." For this case Abaqus allows you to modify the inertia properties of beam and pipe elements, as described below. Loads on the structure associated with incident waves in the fluid can be accommodated under this approximation as well.

Natural Frequency Extraction

Abaqus can compute both real and complex eigensolutions for purely acoustic or structural-acoustic systems, with or without damping. Exterior acoustic problems may also be solved.

Selecting an Eigensolver

In a coupled acoustic-structural model, Abaqus extracts real-valued coupled modes by default using the Lanczos and the subspace iteration eigenfrequency extraction procedures. You can suppress coupling in the frequency extraction step; in this case the structural elements behave as though the interface with the acoustic elements were free (as though this surface were “in vacuo”), and the acoustic elements behave as though the boundary with the structural elements were rigid. You can also extract the coupled acoustic-structural modes when using the AMS eigensolver.

When applying the AMS eigensolver to a coupled structural-acoustic model, Abaqus by default projects and stores the acoustic coupling matrix during the natural frequency extraction, for later use in coupled forced response analyses. The structural and acoustic regions are not actually coupled during the eigenanalysis; Abaqus solves the two regions separately but computes and stores the projected coupling operator for use in subsequent steady-state dynamic steps. Only structural-acoustic coupling defined using tied contact is supported. You can suppress this coupling if desired. Damping due to acoustic volumetric drag is also projected by default during an eigenanalysis and is restored by default in subsequent steady-state dynamic steps. Projecting and storing the acoustic coupling matrix during the natural frequency extraction is also available for the Lanczos and the subspace iteration eigensolvers based on the SIM architecture.

Damping and Inertia Effects in an Acoustic Natural Frequency Extraction

Since damping is not taken into account in real-valued modal extraction, the volumetric drag effect is not considered, except for its small contribution to any nonreflecting boundaries (see *Coupled acoustic-structural medium analysis*). The damping contributions due to any impedance boundary conditions (element-based or surface-based) or acoustic infinite elements are not included in an eigenfrequency extraction step, but the contributions to the acoustic element mass and stiffness matrices are included. Similarly, the (symmetrized) stiffness and mass contributions of acoustic infinite elements are included in an eigenfrequency extraction step, but the damping effects are neglected.

Modal analysis of damped and radiating acoustic systems can be performed in Abaqus as well. Using the complex eigenvalue extraction procedure, the damping contributions of acoustic infinite elements, nonreflecting impedance conditions, and general impedance layers are restored to the element operators.

If an underlying flow field is defined for the acoustic region by specifying an acoustic flow velocity, the natural frequencies and modes are affected. However, in real-valued frequency extraction only the acoustic element mass and stiffness matrices contribute to the solution. Since the formulation for acoustics in the presence of a flow field requires a complex part in the element operator (damping matrix), the real-valued procedure can include the effects of flow only to a limited degree. The complex frequency procedure in Abaqus/Standard includes the damping matrix contribution and is, therefore, required when modes of a system with moving fluid are sought. The complex frequency procedure can be used only following the Lanczos real-valued frequency procedure.

Virtual mass effects defined for beams by adding inertia (*Additional Inertia due to Immersion in Fluid*) are included in modal analysis: their effect is simply to add inertia to a beam element.

Interpreting the Extracted Modes in a Coupled Structural-Acoustic Natural Frequency Analysis

While all the modes extracted in a coupled Lanczos structural-acoustic natural frequency analysis include the effects of fluid-solid interaction, some of them may have predominantly structural contributions while others may have predominantly acoustic contributions. Coupled structural-acoustic eigenmodes can be categorized as follows:

- Most generally, an individual mode may exhibit participation in both the fluid and the solid media; this is referred to as a “coupled mode.”
- Second, there are the “structural resonance” modes. These are modes corresponding to the eigenmodes of the structure without the presence of the acoustic fluid. The presence of the acoustic fluid has a relatively small effect on these eigenfrequencies and the mode shapes.

- Third, there are the “acoustic cavity resonance” modes. These are nonzero eigenfrequency coupled modes that have a significant contribution in the resulting dynamics of the acoustic pressure in mode-based dynamic procedures.
- Fourth, if insufficient boundary conditions are specified on the structural part of a model, the frequency extraction procedure will extract rigid body modes. These modes have zero eigenfrequencies (sometimes they appear as either small positive or even negative eigenvalues). However, if sufficient structural degrees of freedom are constrained, these rigid body modes disappear.
- Finally, there are the singular acoustic modes, which have zero eigenfrequencies and constant acoustic pressure; they are mathematically analogous to structural rigid body modes. The structural part of the singular acoustic modes corresponds to the quasi-static structural response to constant pressure in unconstrained acoustic regions. These eigenmodes are predominantly acoustic and are important in representing the (low-frequency) acoustic response in mode-based analysis in the presence of acoustic loads, in the same way that rigid body modes are important in the representation of structural motion. As is true for the structural rigid body modes, if a sufficient number of constrained acoustic degrees of freedom is specified (one degree of freedom 8 per acoustic region is enough), the singular acoustic modes will disappear. In models with only one unconstrained acoustic region (the most common case) only one singular acoustic mode will be computed. In general there are as many singular acoustic modes as there are independent unconstrained acoustic regions. If these modes are present, they are always reported first by the Lanczos eigensolver; and a note at the bottom of the eigenfrequency table in the data file provides information about the number of singular acoustic modes.

The generalized masses and effective masses can help distinguish between the various types of modes and can be used to assess which modes are important for subsequent mode-based analyses. In addition, the acoustic contribution to the generalized masses is reported as a fraction for each eigenmode. The closer the value of this fraction is to unity, the more pronounced is the acoustic component of this eigenmode. An acoustic effective mass is also computed for each eigenmode. This scalar quantity is scaled such that when all eigenmodes in a model are extracted, the sum of all acoustic effective masses is equal to 1.0 (minus the contributions from nodes with restrained acoustic degrees of freedom). The acoustic effective mass can be compared between different modes: the higher the acoustic effective mass, the more important (typically) the mode is for accurate representation of the acoustic pressure. For example, the fluid cavity acoustic resonance modes will have larger acoustic effective masses compared to the other modes.

Modal Superposition Procedures

In Abaqus acoustic domains are handled quite similarly to solid and structural domains. Real-valued eigenmodes, resulting from a previous real-valued eigenfrequency extraction procedure with or without coupling effects included, are used as a basis for modal solutions. The mode-based steady-state dynamic procedure is the most computationally efficient alternative to compute the steady-state response of structural-acoustic systems. Structural-acoustic coupling and damping effects in these analyses depend on the type of modal procedure and the eigensolver that was used to compute the eigenfrequencies.

Structural-Acoustic Coupling in Modal Analyses Using the Lanczos Eigensolver without the SIM Architecture

If coupled modes are computed using the Lanczos eigensolver, both the mode-based and subspace projection steady-state dynamic procedures will include structural-acoustic coupled effects. If uncoupled Lanczos modes are computed, coupling can be restored only by using subspace projection. It is sufficient to project at a single frequency (constant subspace) to resolve the acoustic coupling for all frequencies.

Acoustic Medium Damping in Modal Analyses Using the Lanczos Eigensolver without the SIM Architecture

In subspace-based steady-state dynamic analysis, acoustic medium damping and structural material damping are considered, and the structural-acoustic interaction, infinite element, and impedance boundary terms are also included.

Acoustic medium damping is not considered in the procedures that base the response prediction directly on the system's eigenmodes, such as transient modal dynamic analysis or the mode-based steady-state dynamic procedure. These methods should, therefore, be used with caution for problems with impedance boundary conditions. Modal damping can be used in these procedures (*Material Damping*) to model material damping and volumetric drag effects; however, modal damping usually cannot be used to model the fluid-solid coupling or the impedance boundary effects accurately.

Structural-Acoustic Coupling and Damping in Modal Analyses Using the Subspace Iteration Eigensolver

The subspace iteration eigensolver neglects the effects of structural-acoustic coupling; therefore, coupling effects are not included in subsequent modal procedures.

As with analyses using the Lanczos eigensolver, acoustic medium damping and structural material damping are considered in subsequent subspace-based steady-state dynamic procedures, but these damping effects are not considered in subsequent transient modal or mode-based steady-state dynamic procedures.

Structural-Acoustic Coupling and Damping in Modal Analyses Using the AMS Eigensolver or the Lanczos Eigensolver Based on the SIM Architecture

The structural-acoustic modes, extracted using the AMS eigensolver or the Lanczos eigensolver, can be used in modal analyses using the SIM architecture. When uncoupled modes are computed using the AMS eigensolver or the Lanczos eigensolver based on the SIM architecture with projection of the structural-acoustic coupling specified, the coupling and acoustic damping operators are projected and stored during the natural frequency extraction. Subsequent coupled forced response analyses using modal steady-state dynamics automatically restore the effects of structural-acoustic coupling and damping by automatically using these projected matrices; if the matrices were not projected, the steady-state dynamic step would not include these effects. A mode-based steady-state dynamic step cannot use unsymmetric damping, such as from acoustic flow velocity or infinite element effects. To take these effects into account, a subspace-based steady-state dynamic analysis should be used.

Defining Translational or Rotational Underlying Flow Velocity in Abaqus/Standard

As described above, acoustic analysis in Abaqus/Standard can be performed as a linear perturbation of a high-speed flow field. The flow velocity field affects the propagation of acoustic waves in the medium through the effect of the flow velocity on the speed of the wave propagation. Waves travel faster along the direction of the local flow vector and are correspondingly impeded in the direction against the flow direction. It is sufficient for you to define the velocity field in the affected acoustic region; the accelerations do not play a role in the formulation.

You specify the flow in the acoustic finite element region as history data within a dynamic linear perturbation step. The flow field can be described either by direct input of the velocity components or by defining rotating motion associated with a reference frame. In the former case, each node in the acoustic region where flow occurs is assigned a Cartesian velocity defined by specifying the components of the velocity vector, $\mathbf{c}$. In the latter case, the rotational velocity for the nodes in the acoustic region is defined by specifying the magnitude of an angular rotation velocity, $\boldsymbol{\Omega}$, and the position and orientation of the axis of rotation in the current configuration. The position and orientation of the axis are applied at the beginning of the step and remain fixed during the step.

The specified underlying flow is active only for acoustic finite elements; other elements with acoustic degrees of freedom, such as acoustic infinite and interface elements, are unaffected by the specified flow velocity. The

effect of underlying flow on the acoustic finite elements depends also on the procedure used: the effects are present only in frequency-domain dynamic procedures and natural frequency extraction. For complex-valued procedures, such as complex frequency extraction and steady-state dynamics, the presence of underlying flow affects the acoustic finite element stiffness matrices and adds a significant contribution to the element damping matrix. For real-valued procedures, such as eigenfrequency extraction and steady-state dynamics analysis in which a real-only system matrix is factored, the presence of underlying flow affects only the acoustic finite element stiffness matrices; the damping matrix is ignored. Consequently, the effect of flow on the acoustic field is fully realized only in complex-valued procedures.

For rotating systems, solid and acoustic materials are treated differently in Abaqus. Flow of solid material through a mesh may induce significant deformation and is handled by using steady-state transport; subsequent linear perturbation steps are analyzed about this deformed state (see *Steady-State Transport Analysis*). Flow of material through an acoustic mesh is handled entirely within linear perturbation steps by specifying an acoustic flow velocity; a preliminary nonlinear steady-state transport analysis is not required. For coupled acoustic-structural systems undergoing rotation, such as tires, the model may be subjected to a steady-state transport step, which deforms the solid medium, followed by linear perturbation dynamic steps. The effect of the rotation of the solid is included in the linear perturbation steps in this case; to include the effect of the rotation of the acoustic fluid, specify an acoustic flow velocity in the linear perturbation steps.

Input File Usage: Use the following option to define a translating fluid velocity:

**ACOUSTIC FLOW VELOCITY*, TRANSLATION

Use the following option to define a rotating fluid velocity:

**ACOUSTIC FLOW VELOCITY*, ROTATION

Abaqus/CAE Usage: Acoustic flow velocity is not supported in Abaqus/CAE.

Updating the Acoustic Domain during a Large-Displacement Abaqus/Standard Analysis

By default, the acoustic-structural coupling calculations are based on the original configuration of the fluid domain. However, acoustic elements can also be used in an analysis where the structural domain experiences large deformation prior to the coupled analysis. A typical example is the interior cavity of a tire subjected to structural loads such as inflation, rim mounting, and footprint pressure.

The acoustic elements in Abaqus do not have displacement degrees of freedom and, therefore, cannot model the deformation of the fluid when the structure undergoes large deformation. Abaqus/Standard solves the problem of computing the current configuration of the acoustic domain by periodically creating a new acoustic mesh. The new mesh uses the same topology (elements and connectivity) throughout the simulation, but the nodal locations are adjusted so that the acoustic domain conforms to the structural domain on the boundary.

A new acoustic mesh is computed when adaptive meshing is specified and nonlinear geometric effects are considered in any non-perturbation Abaqus/Standard analysis procedure in which acoustic effects are ignored.

The adaptive meshing features for acoustic analysis are described in detail in *Defining ALE Adaptive Mesh Domains in Abaqus/Standard* and *ALE Adaptive Meshing and Remapping in Abaqus/Standard*.

Initial Conditions

In Abaqus/Standard the initial acoustic static pressure (hydrostatic or ambient) is not modeled by the acoustic formulation and cannot be specified as an initial condition.

In Abaqus/Explicit the initial acoustic pressure corresponding to the initial static equilibrium (hydrostatic or ambient) can be specified (see *Initial Conditions*) and is meaningful only when the acoustic fluid is capable of

undergoing cavitation. In problems with possible fluid cavitation the initial acoustic static pressure is taken into account in the cavitation condition; i.e., the sum of the dynamic and static acoustic pressures needs to drop to the cavitation pressure limit for the cavitation to occur. The specified acoustic static pressure is used only in the cavitation condition and does not apply any static loads to the acoustic or structural meshes at their common wetted interface. In addition, the acoustic static pressure is not included in the nodal acoustic pressure degree of freedom.

The initial temperature and field variable values can be specified (*Initial Conditions*) for the direct time integration dynamic, explicit dynamic, dynamic fully coupled temperature-displacement, and mode-based transient dynamic analysis procedures. Changes in these variables during the analysis will affect any temperature-dependent or field-variable-dependent acoustic medium properties.

Boundary Conditions

The various boundary conditions that can be applied to an acoustic medium are described below. These include acoustic domain boundaries with stationary rigid walls or symmetry planes, prescribed pressure values such as a free surface with zero dynamic pressure, specified impedance (see *Acoustic and Shock Loads*), and structural interfaces such as the interface with a ship or a submarine. The radiating (nonreflecting) boundary condition for exterior problems (such as a structure vibrating in an acoustic medium of infinite extent) is implemented as a special case of the impedance boundary condition (see *Acoustic and Shock Loads*). On any given part of the acoustic domain boundary only one boundary condition type should be applied, except for the combination of the impedance boundary condition and the acoustic-structural interface condition.

Boundary with a Stationary Rigid Wall or a Symmetry Plane

The default boundary condition for an acoustic medium is a boundary with a stationary rigid wall or a symmetry plane. The “force” conjugate to pressure in the acoustics formulation in Abaqus is the normal pressure gradient at the surface divided by the mass density; dimensionally this is equal to a force per unit mass. In the absence of volumetric drag this force per unit mass is equal to the inward acceleration of the acoustic medium. The conjugate variable at a node on the surface is the inward volume acceleration, which is the integral of the inward acceleration of the acoustic medium evaluated over the surface area associated with the node. A “traction-free” surface (one with no boundary conditions, no applied loads, no surface impedance conditions, and no interface elements) is a surface normal to which the acoustic medium undergoes no motion and, thus, corresponds to a rigid, stationary surface adjacent to the fluid. A symmetry plane for the acoustic medium is another “traction-free” surface.

Prescribed Pressure

The basic variable in the acoustic medium is pressure (degree of freedom 8). Therefore, this variable can be prescribed at any node in the acoustic model by applying a boundary condition (*Boundary Conditions*). Setting the pressure to zero represents a “free surface,” where the pressure does not vary because of the motion of the surface (to account for surface motion effects, see the discussion of impedance below). Prescribing a nonzero value for the pressure represents a sound source.

An amplitude variation can be used to specify the value of the pressure. In a steady-state analysis you can specify both the in-phase (real) part of the pressure (default) and the out-of-phase (imaginary) part of the pressure.

Input File Usage: Use either of the following options to define the real (in-phase) part of the boundary condition:

**BOUNDARY*
**BOUNDARY*, REAL

Use the following option to define the imaginary (out-of-phase) part of the boundary condition:

**BOUNDARY*, IMAGINARY

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Acoustic pressure** for the **Types for Selected Step**: select regions: **Magnitude**: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**

Boundary with a Structure

If the acoustic medium is adjacent to a structure, there will be a transfer of momentum and energy between the media at the boundary. The pressure field modeled with acoustic elements creates a normal surface traction on the structure, and the acceleration field modeled with structural elements creates the natural forcing term at the fluid boundary (for details, see *Coupled acoustic-structural medium analysis*).

The surface-based coupling procedure and the user-defined acoustic interface elements differ slightly in their theoretical implementation. In essence, the interface elements computed internally by the surface-based procedure are discrete point elements computed at the nodes of the secondary surface. A user-defined acoustic interface element, on the other hand, distributes coupling effects across all of its nodes. Generally, the results obtained using the two coupling methods will be very similar, but the difference in discretization at the coupling boundary may create small differences in results.

Defining Acoustic-Structural Coupling with User-Defined Acoustic Interface Elements

In Abaqus/Standard, if the structural and acoustic meshes share nodes at the boundary, lining this boundary with acoustic-structural interface elements (see *Acoustic Interface Element Library*) will enforce the required physical coupling condition. The interface element normals must point into the acoustic medium, which forces continuity of the normal accelerations of the acoustic medium and of the structure at the boundary and ensures that the pressure of the acoustic elements is applied to the structure. Displacements can also be prescribed at such a boundary.

Defining Acoustic-Structural Coupling Using a Surface-Based Coupling Procedure

Alternatively, a surface-based procedure can be used to enforce the coupling; in Abaqus/Explicit the surface-based procedure is the only available method. This method requires that the structural and acoustic meshes use separate nodes. You define surfaces on the structural and fluid meshes and define the interaction between the two meshes using a surface-based tie constraint (see *Mesh Tie Constraints*). No additional element definitions are required.

The secondary surface, the first of the two surfaces specified for the tie constraint, must be element-based; whereas the main surface can be either element- or node-based. A surface based on rigid element types (R3D4, etc.) or an analytical rigid surface cannot be used as a main surface; instead, use a deformable surface made rigid.

For surface-based tie constraints Abaqus automatically computes the region of influence for each internally generated acoustic-structural interface element. If the user-defined secondary surface overhangs the main surface significantly, the regions of influence may include parts of the overhang. Consequently, the overhanging part of the secondary surface may exhibit nonphysical coupled degrees of freedom: displacements if the secondary surface is acoustic and acoustic pressures if the secondary surface is solid or structural. These nonphysical results on the overhang do not affect the remainder of the solution, and it should be understood that they are not meaningful.

Input File Usage: Use the following option in an analysis with the fluid mesh surface as the secondary:

**TIE*, NAME=*fluidsecondary*
fluid_surf, struct_surface

Use the following option in an analysis with the solid mesh surface as the secondary:

**TIE*, NAME=*solidsecondary*
struct_surf, fluid_surf

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**

Coupling Surfaces to Structures Using Acoustic Infinite Elements

Acoustic infinite elements may form surfaces that can be coupled to structures by using a tie constraint in two different ways. The acoustic infinite element surface may consist of the base (first) facets of the acoustic infinite elements; in this case this surface should be tied to a topologically similar structural surface. The acoustic infinite element edges may also be used to define surfaces (see *Mesh Tie Constraints*), which can be tied to solid elements. This approach couples the semi-infinite sides of acoustic infinite elements to solid elements.

Mesh Refinement

Although the meshes may be nodally nonconforming at the tied surfaces, mesh refinement affects the accuracy of the coupled solution. In acoustic-solid problems the mesh refinement depends on the wave speeds in the two media. The mesh for the medium with the lower wave speed should generally be more refined and, therefore, should be the secondary surface. If the details of the wave field in the vicinity of the fluid-solid interface are important, the meshes should be of equally high refinement, with the refinement corresponding to the lower wave speed medium. In this case the choice of the main surface is arbitrary. An exception is the case where the acoustic medium must be updated to follow the structure during a large-deformation analysis. In such a case the secondary surface must be defined on the acoustic domain. Another exception is the case of fluids coupled to both sides of shell or beam elements (as described below).

Solving the Structural System Sequentially Using the Submodeling Technique

In some applications the normal surface traction on the structure created by the acoustic fluid may be negligible compared to other forces in the structural system. For example, a metal motor housing may radiate sound into the surrounding air, but the reaction pressure of the air on the motor may be insignificant to the dynamics of the housing. In these cases the submodeling technique (see *About Submodeling*) can be used to solve the system sequentially; that is, the structural analysis (uncoupled from the fluid) is followed by the acoustic analysis (driven by the structure). Usually, this decoupling of the analysis reduces computational cost. The structural system plays the role of the “global” model, and the acoustic fluid is the submodel. The structural displacements on the boundary of the acoustic fluid must be saved to the results file in the global analysis. Since Abaqus interpolates the fields between the global and submodels, acoustic-structural interface elements can be used. They should be applied to the fluid boundary to be driven by the global structural model.

Input File Usage: Use the following options in the global (structural) analysis to be followed by a submodeling analysis:

**NSET*, NSET=*solid_boundary_driving_the_fluid*
**NODE FILE*, NSET=*solid_boundary_driving_the_fluid*
U

Use the following options in the subsequent submodeling (fluid) analysis that uses acoustic interface elements on the fluid boundary to be driven:

**NSET*, NSET=*fluid_boundary_to_be_driven*
**SUBMODEL*, EXTERIOR TOLERANCE=*tolerance*
fluid_boundary_to_be_driven
**BOUNDARY*, SUBMODEL, STEP=1
fluid_boundary_to be_driven, 1, 3,

Abaqus/CAE Usage: Use the following input in the submodeling (fluid) analysis that uses an acoustic interface on the fluid boundary to be driven:

Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**:
select regions for *fluid_boundary_to _be_driven*: **Exterior tolerance**: **relative**: *tolerance*; **Degrees of freedom**: 1, 3; **Global step number**: 1

Defining Acoustic-Structural Coupling on Both Sides of a Beam or Shell

In Abaqus/Standard there are two alternatives available for modeling a beam (in two dimensions) or shell interacting with fluid on both sides: a surface-based procedure and an element-based procedure. In Abaqus/Explicit the surface-based procedure must be used.

Use of the surface-based procedure is straightforward. Two surfaces must be defined on the beam or shell: one on the SPOS side and one on the SNEG side. Each surface is then coupled to the fluid using a tie constraint. At least one of the two surfaces on the beam or shell must be a main surface.

In Abaqus/Standard, if the same nodes are used for the fluid and the beam or shell, acoustic interface elements must be used in the following manner to define acoustic-structural coupling on both sides of a beam or shell element:

1. Define a second set of nodes coincident with the beam or shell nodes, and constrain the motions of the two sets of nodes together using a PIN-type MPC (*General Multi-Point Constraints*).
2. Use the first set of nodes to line one side of the beam or shell elements with acoustic interface elements (with the normals of the acoustic interface elements pointing into the fluid).
3. Use the second set of nodes to line the other side of the beam or shell elements with acoustic interface elements (with the normals pointing into the fluid on the opposite side of the structure, as in Step 2).
4. The acoustic elements on the first side of the beam or shell elements should be defined using the first set of nodes, and the acoustic elements on the second side of the beam or shell elements should be defined using the second set of nodes.

Defining the Virtual Mass Effect (Fluid-Structural Coupling) for Beam and Pipe Elements

In Abaqus virtual mass effects on submerged Timoshenko beam elements can be modeled by specifying additional inertia for the beam. The virtual mass effects are specified as part of the section definition of the beam.

1. Define the beam section (*Using a Beam Section Integrated during the Analysis to Define the Section Behavior* or *Using a General Beam Section to Define the Section Behavior*), any additional internal inertia (*Adding Inertia to the Beam Section Behavior for Timoshenko Beams*), and the beam material properties.
2. Define the virtual mass effect (*Additional Inertia due to Immersion in Fluid*).
3. If the model is to be loaded using an incident wave (*Incident Wave Loading due to External Sources*), define a surface or surfaces on the beam elements.

Loads

The following types of loading can be prescribed in an acoustic analysis, as described in *Acoustic and Shock Loads*:

- Concentrated pressure-conjugate loading.
- An impedance condition that specifies the relationship between the pressure of the acoustic medium and the normal motion at the boundary (either element-based or surface-based). Such a condition is applied, for example, to include the effect of small-amplitude “sloshing” in a gravity field or to include the effect of a compressible, possibly dissipative, lining (such as a carpet) between the acoustic medium and a fixed, rigid wall or a structure. This type of condition can also be applied to edge facets of acoustic infinite elements.
- Nonreflecting radiation conditions on acoustic boundaries (either element-based or surface-based). An impedance can be defined to select the appropriate radiating boundary condition taking the radiating surface shape into consideration.
- Incident wave loading such as that caused by an underwater explosion or a sound field. Since this type of loading is usually applied in conjunction with semi-infinite acoustic regions, two alternative modeling formulations are available in Abaqus. A total pressure-based formulation is provided when the incident wave loading is applied to the exterior of a semi-infinite acoustic mesh. This formulation must be used to handle the incident wave loading when the acoustic medium is capable of cavitation, rendering the fluid material behavior nonlinear. The scattered pressure formulation is typically used when cavitation is not part of the fluid mechanical behavior and when the loads are applied to fluid-solid interfaces. Sound transmission loss and acoustic scattering problems usually fall into the latter category.

  For both formulations, when incident wave loading is applied to a given surface, a mathematical jump occurs between the pressures on both sides of the surface because the side from which the incident pressure arrives is implicitly a region of scattered pressure. This jump is handled automatically when the incident wave load is applied to a surface with a nonreflecting impedance condition and when the incident wave load is applied to a fluid-solid interface. However, if the incident wave load is applied to a surface lying between acoustic finite or infinite elements, the jump will not be modeled properly because pressures are continuous between acoustic elements. For this case, low-mass and low-stiffness membrane, shell, or surface elements should be interposed between the acoustic elements to permit the jump in pressure to exist.

  Incident wave loading can be applied in time-harmonic problems, using the direct solution steady-state dynamics and the subspace-based, steady-state dynamic procedures. You can define individual spherical or planar sources emitting waves, or you can use the deterministic diffuse field model in Abaqus. In the former case, usage is quite similar to transient analysis: the defined sources correspond to distinct sound sources. The latter case models the sound field incident on a surface exposed to a reverberant chamber: the field is assumed to be equivalent to a number of plane waves arriving from directions distributed on a hemisphere. Only the scattered wave formulation is supported when using incident wave loading in steady-state dynamics.

  See *Acoustic and Shock Loads* for several examples of incident wave loading.
- Loading due to an incident shock wave caused by an air explosion. Although this type of wave is highly nonlinear and complex, the pressure loading due to the shock wave can be calculated readily from empirical data provided by the CONWEP model available in Abaqus/Explicit. The main advantage of this model is that the loading is applied directly to the structure subject to the blast; there is no need to include the fluid medium in the computational domain. In the CONWEP model, empirical data for two types of waves are available: spherical waves for explosions in mid-air and hemispherical waves for explosions at ground level in which ground effects are included.

  The CONWEP model does not account for effects of shadowing by intervening objects. In addition, it does not account for effects due to confinement and, thereby, excludes incorporation of any reflecting surfaces in the analysis. The model does account for the angle of incident of the blast wave; see *Acoustic and Shock Loads* for incorporation of the incident angle in the pressure load calculation.

Predefined Fields

The following predefined fields can be specified in an acoustic analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in acoustic elements, nodal temperatures can be specified. The specified temperature affects temperature-dependent material properties.
- The values of user-defined field variables can be specified. These values affect field-variable-dependent material properties.

Material Options

Only the acoustic medium material model (*Acoustic Medium*) is valid for use in an acoustic analysis. The structure in a coupled acoustic-structural analysis can be modeled using any material model. Since acoustic analyses are always performed using a dynamic procedure, the structure's density (*Density*) should usually be defined.

Porous materials are often modeled using an acoustic formulation when the dilatational waves in the porous medium dominate the shear effects. A large number of models exist for this category of phenomenon. In Abaqus, two categories of models are available for porous media modeled with acoustic elements: phenomenological models and general frequency-dependent models. Phenomenological models describe the dynamic characteristics using material data related to the porous structure, such as porosity itself, tortuosity, etc. Alternatively, you can specify the dynamic properties directly for the material; usually, this is done using a table of frequency-dependent data. See *Acoustic Medium* for details on specifying acoustic materials in Abaqus.

When the acoustic medium is capable of cavitation and the analysis includes incident wave loading, a total pressure-based formulation must be used. Either the default scattered wave formulation or the total wave formulation can be used if the cavitation is absent or the problem has no incident wave loading.

For beam elements using the virtual mass approximation, the relevant data are specified as part of the beam section definition.

Elements

Abaqus provides a set of elements for modeling an acoustic medium undergoing small pressure changes. In addition, Abaqus/Standard provides interface elements to couple these acoustic elements to a structural model (see *Choosing the Appropriate Element for an Analysis Type*). If interface elements are used, only direct-solution steady-state harmonic (linear) response analysis (*Direct-Solution Steady-State Dynamic Analysis*) and transient response analysis (*Implicit Dynamic Analysis Using Direct Integration*) can be performed.

In Abaqus/Standard the second-order acoustic elements are generally considerably more accurate than first-order acoustic elements for a given number of degrees of freedom. The acoustic elements in Abaqus/Explicit are limited to first-order interpolations.

Acoustic elements cannot be used together with hydrostatic fluid elements.

With the CONWEP model provided in Abaqus/Explicit, the analysis must be three-dimensional. The loading surface must be comprised of solid, shell, or membrane elements only. In addition, CONWEP loading cannot be applied to acoustic elements.

Exterior Problems

We often need to model an exterior problem, such as a structure vibrating in an acoustic medium of infinite extent. Impedance-type radiation boundary conditions can be used to model the motions of waves out of the mesh. Abaqus provides acoustic infinite elements for this class of problems. In addition, Abaqus/Standard provides perfectly matched layers to truncate the acoustic infinite domain in a direct-solution steady-state dynamic analysis.

Impedance-Type Radiation Conditions

In this case acoustic elements are used to model the region between the structure and a simple geometric surface (located away from the structure), and a radiating (nonreflecting) boundary condition is applied at that surface. The radiating boundary conditions are approximate, so that the error in an exterior acoustic analysis is controlled not only by the usual finite element discretization error but also by the error in the approximate radiation condition. In Abaqus the radiation boundary conditions converge to the exact condition in the limit as they become infinitely distant from the radiating structure. In practice, these radiation conditions provide accurate results when the distance between the surface and the structure is at least one-half of the longest characteristic or responsive structural wavelength.

For details, see *Acoustic and Shock Loads*.

Acoustic Infinite Elements

Acoustic infinite elements are provided for modeling exterior problems (*Infinite Elements*). These elements have surface topology: line and quadratic segments in two-dimensional and axisymmetric problems and triangles and quadrilaterals in three-dimensional problems. Generally, the acoustic infinite elements are defined on a terminating surface of a region of acoustic finite elements. The infinite element formulation is considerably more accurate than the impedance-type radiation boundary conditions in cases where the wave field impinging on the terminating surface has many complex features. The radiation boundary conditions are relatively simple, equivalent to a "zeroth-order" infinite element; the acoustic infinite elements implemented in Abaqus are of variable order, up to ninth.

Acoustic infinite elements can be coupled directly to structural surfaces by using a surface-based tie constraint: this may provide adequate accuracy in some applications. In general cases the acoustic infinite elements are defined on the terminating surface of the acoustic finite element mesh. The diameter of the acoustic finite element mesh can be considerably smaller, for a given solution accuracy, than is the case when using radiation boundary conditions.

The nodal connectivity on the acoustic infinite element defines the element's surface topology. To complete the element formulation, the surface topology must be mapped into the infinite domain. This mapping requires a reference point, given in the element section property definition. The reference point serves to define a characteristic length used in the coordinate mapping. In the ideal case of acoustic radiation from a spherical surface, the correct placement of the reference point is the center of the sphere. In general, the acoustic infinite elements produce the most accurate results when the reference node is located near the center of the region enclosed by the infinite elements.

Nodal normal vectors are required for an accurate mapping of the infinite domain. The nodal normal vectors must point into the infinite domain and are used to define the portion of the infinite domain treated by a particular infinite element. To cover the infinite domain without overlap, each node attached to an infinite element must have a unique normal. The nodal normal vectors are specified or calculated as follows.

User-specified alternative nodal normals (*Normal Definitions at Nodes*) are ignored for acoustic infinite elements and, therefore, cannot be used to define normal directions for acoustic elements. Over the element's surface topology, the normal vectors must be divergent; that is, the area mapped (in two dimensions) or the volume mapped (in three dimensions) must increase with distance into the infinite domain. To ensure this criteria, the normal vectors at each acoustic infinite element node are defined to lie along the vector between that node and the reference point given in the element section property definition. See *Infinite Elements* for more information.

Perfectly Matched Layers

Perfectly matched layers are provided for modeling exterior acoustic problems during a direct-solution steady-state dynamic analysis. The perfectly matched layer is modeled using the acoustic elements that are available in Abaqus/Standard whose behavior is modified appropriately to act as an absorbing layer. The perfectly matched layer absorbs all the waves that are incident to it. For any given problem, it is recommended to have 4–7 layers of elements in the perfectly matched layer region. You specify the coefficients that are used to define the absorbing

properties of the perfectly matched layer and the outer limits of the acoustic domain. These limits are used to compute the absorbing properties for the perfectly matched layer.

You cannot use axisymmetric acoustic elements to model the perfectly matched layer. The outer surface of the perfectly matched layer should have a zero pressure boundary condition. To avoid reflections from the perfectly matched layer region, the material properties used for the perfectly matched layer elements should match those of the acoustic elements.

Input File Usage: **PERFECTLY MATCHED LAYER*, ELSET=*element set label*, MATERIAL=*material name*
**PML COEFFICIENT*, VARIATION=LINEAR

Abaqus/CAE Usage: You cannot define a perfectly matched layer property in Abaqus/CAE.

Mesh Refinement

Inadequate mesh refinement is the most common source of difficulties in acoustic and vibration analysis. For reasonable accuracy, at least six representative internodal intervals of the acoustic mesh should fit into the shortest acoustic wavelength present in the analysis; accuracy improves substantially if ten or more internodal intervals are used at the shortest wavelength. In steady-state analyses the shortest wavelength will occur in the medium with the lowest speed of sound, at the highest frequency analyzed. In transient analyses the shortest wavelength present is more difficult to determine before an analysis: it is reasonable to estimate this wavelength using the highest frequency present in the loads or prescribed boundary conditions.

An “internodal interval” is defined as the distance from a node to its nearest neighbor in an element; that is, the element size for a linear element or half of the element size for a quadratic element. At a fixed internodal interval, quadratic elements are more accurate than linear elements. The level of refinement chosen for the acoustic medium should be reflected in the solid medium as well: the solid mesh should be sufficiently refined to accurately model flexural, compressional, and shear waves.

The level of mesh refinement required depends on the application. Any finite element discretization of a domain in which waves propagate introduces a certain amount of error per wavelength. In meshes that are small in terms of wavelengths, relatively coarse (for example, six internodal intervals per wavelength) meshes may be adequate. For meshes that contain many wavelengths at the frequency of interest, the per-wavelength finite element discretization error accumulates, generally necessitating greater levels of refinement. In these larger meshes the accumulated per-wavelength error may be present throughout the mesh if refinement is inadequate.

The acoustic wavelength decreases with increasing frequency, so there is an upper frequency limit for a given mesh. Let L_{max} represent the maximum internodal interval of an element in a mesh, n_{min} the number of internodal intervals we desire per acoustic wavelength ($n_{min} \geq 10$ is recommended), f_{max} the cyclical frequency of excitation, and $c = \sqrt{\mathrm{K}_f/\rho_f}$ the speed of sound, where K_f is the bulk modulus of the acoustic medium and ρ_f is its density. The requirements are then expressed as

$$L_{max} < \frac{c}{n_{min} f_{max}} \qquad \text{or} \qquad f_{max} < \frac{c}{n_{min} L_{max}}.$$

The above expressions can be used to estimate the maximum allowable element length if the frequency is given or the maximum frequency for which a given mesh size is valid. For example, in air at room temperature, $c \approx 343$ meters per second. The following table gives some values for maximum internodal distances to model given maximum frequencies f_{max} accurately:

Maximum Frequency of Interest, f_{max}	Maximum Internodal Interval, L_{max}, $n_{min} \equiv 8$	Maximum Internodal Interval, L_{max}, $n_{min} \equiv 12$
100 Hz	< 430 mm	< 286 mm
500 Hz	< 86 mm	< 57 mm
1000 Hz	< 43 mm	<29mm
20 kHz	< 2.1 mm	< 1.4mm

For exterior problems the accuracy of an analysis also depends on the accuracy of the absorbing boundary condition. As mentioned above, the absorbing boundary impedance conditions implemented in Abaqus are used with a standoff thickness r_1 of acoustic finite elements between the acoustic sources and the radiating boundary. Since the approximate radiation conditions converge to the exact condition in the limit of infinite standoff, a greater standoff thickness improves the accuracy of the solution. The standoff thickness r_1 is expressed as m_{min} wavelengths at the minimum frequency to be analyzed:

$$r_1 > \frac{c\, m_{min}}{f_{min}} \qquad \text{or} \qquad f_{min} > \frac{c\, m_{min}}{r_1}.$$

Continuing the example using the properties of air, we can calculate the recommended minimum standoff thicknesses corresponding to a specified minimum frequency of interest, using $m_{min} = 1/3$:

Minimum Frequency of Interest, f_{min}	Radiation Boundary Standoff, r_1
100 Hz	> 1140 mm
500 Hz	> 230 mm
1000 Hz	> 114 mm
20 kHz	> 5.7 mm

The computational requirements for an exterior problem thus depend on both the radiation boundary standoff and the internodal distance. The number of nodes *N* in a model depends on the volume of the mesh, controlled by r_1 and the spatial dimension *d*, and the mesh density, controlled by L_{max}. The exact number of nodes depends on the details of the model, but the expression

$$N \sim \left(\frac{r_1}{L_{max}}\right)^d \sim \left(\frac{f_{max}}{f_{min}}\, m_{min}\, n_{min}\right)^d$$

indicates the size of the model with respect to the ratio of the maximum to minimum frequencies in a given analysis. Because the mesh size for an exterior problem exhibits such strong dependence on the bandwidth, $f_{max} - f_{min}$, you can control the size of an analysis by splitting the band. For example, if the overall frequency range of interest is 100 to 10000 Hz, a single spherical mesh covering this band in three dimensions has size

$$N \sim (100\, m_{min}\, n_{min})^3 \sim 1e6 (m_{min}\, n_{min})^3.$$

However, splitting the problem into two bands, $(100,\ 1000)$ and $(1000,\ 10000)$, and creating an exterior mesh for each band, results in two analyses of size

$$N \sim (10\, m_{min}\, n_{min})^3 \sim 1000\, (m_{min}\, n_{min})^3.$$

In coupled acoustic-structural systems there usually exist different wave speeds for the fluid and solid media. In the region of the acoustic-structural interface, the wave phenomena in both media may exhibit length scales characteristic of the slower medium; that is, the length scale of the wave dynamics may be as short as the shorter wavelength, corresponding to the lower wave speed. This result follows from the fact that the two media are coupled at the boundary. The region near the acoustic-structural interface where these effects are important is usually no thicker than the shorter wavelength.

For example, in an analysis involving water interacting with rubber, the wave speed in the rubber may be much lower than that of water. A finite element mesh used to model this problem in detail would require refinement down to six (or more) nodes per shorter wavelength, on both sides of the interface. On the water side (faster, longer wavelength) accuracy will probably not be compromised significantly if this region of high refinement extends no further into the water than one short wavelength. Of course, in some analyses the effects in the vicinity of the interface may be unimportant. Then, the two meshes can be refined only so far as to represent their own characteristic wavelengths accurately.

Output

Nodal output variable PAC (pressure magnitude at the nodes of the acoustic elements, alias of POR) is available for an acoustic medium (in Abaqus/CAE this output variable is called POR). When the scattered wave formulation (default) is used with incident wave loading, output variable POR represents only the scattered pressure response of the model and does not include the incident wave loading itself. When the total wave formulation is used, output variable POR represents the total dynamic acoustic pressure, which includes contributions from both incident and scattered waves as well as the dynamic effects of fluid cavitation. For either formulation output variable POR does not include the acoustic static pressure.

In Abaqus/Explicit an additional nodal output variable PABS (the absolute pressure, equal to the sum of POR and the acoustic static pressure) is available. When the dynamic effects of fluid cavitation are of interest, you can specify the acoustic static pressure in an acoustic analysis that uses the total wave formulation. If the acoustic static pressure is not specified in an acoustic region, it is assumed to be large; thus precluding cavitation in that region.

For general steps, including implicit and explicit dynamic steps, no energy quantities are computed for acoustic elements. Consequently, these elements will not contribute to the total energy balance.

Steady-State Dynamic Output

For steady-state dynamic analysis POR is complex and can be displayed in several forms in the Visualization module of Abaqus/CAE. The phase angle (PPOR) is available as output to the data (`.dat`) and results (`.fil`) files.

Several additional secondary quantities are available for multidimensional acoustic finite elements in direct-solution steady-state dynamic or subspace-based steady-state dynamic analysis. The “sound pressure level” is defined as:

$$SPL \equiv 20 \log_{10} p_{RMS} - 20 \log_{10} p_{REF},$$

where p_{REF} is defined as a physical constant in the model (see *Defining the Reference Pressure* below), and the p_{RMS} is computed from the complex-valued acoustic pressure $\tilde{p}$ at any point using the formula:

$$2p_{RMS}^2 = \Re(\tilde{p})^2 + \Im(\tilde{p})^2.$$

The acoustic particle velocity at any material point is

$$\tilde{\mathbf{v}} = \frac{i}{\tilde{\rho}\omega} \frac{\partial \tilde{p}}{\partial \mathbf{x}}.$$

The acoustic intensity vector, a measure of the rate of flow of energy at a material point, is

$$\tilde{\mathbf{I}} = -\frac{1}{2} \sigma \hat{\tilde{\mathbf{v}}}.$$

In an acoustic medium the stress tensor is simply the acoustic pressure times the identity tensor, so this expression simplifies to

$$\tilde{\mathbf{I}} = \frac{-1}{2i\hat{\tilde{\rho}}\omega}\left[\tilde{p}\frac{\partial\hat{\tilde{p}}}{\partial \mathbf{x}}\right].$$

The hats denote complex conjugation. The real part of the intensity is referred to as the "active intensity," and the imaginary part is the "reactive intensity." The acoustic pressure gradient is also available for acoustic finite elements in steady-state dynamic analysis.

In steady-state dynamic analysis, additional nodal output quantities are available for acoustic infinite elements.

PINF denotes the complex pressure coefficients of the infinite element shape functions. These coefficients can be used to visualize the exterior acoustic field (i.e., within the volume of the acoustic infinite elements) using scripting in the Visualization module of Abaqus/CAE; see *Using infinite elements to compute and view the results of an acoustic far-field analysis*. INFN is the normal vector used by the acoustic infinite element to define the element volume. INFR denotes the radius used for the element at that node, and INFC denotes the element cosine; that is, the minimum dot product between the nodal normal vector and the acoustic infinite element facet normal vectors attached to that node. See *Acoustic infinite elements* for more complete descriptions of these quantities. INFN, INFR, INFC are useful in debugging a model using acoustic infinite elements; consequently, it is sometimes valuable to perform a steady-state dynamics, direct analysis on a model to visualize this information.

For steady-state dynamic steps, energy quantities are available for acoustic elements. These elements contribute to the total energy balance in steady-state dynamics.

Defining the Reference Pressure

You must define the reference pressure, p_{REF}, used to compute the sound pressure level; there is no default value for the reference pressure.

Input File Usage: **PHYSICAL CONSTANTS*, SPL REFERENCE PRESSURE=p_{REF}

Abaqus/CAE Usage: You cannot define a reference pressure in Abaqus/CAE.

Input File Template

The following is an example of the step definition for a direct-solution steady-state dynamic acoustic analysis that looks for the response of a model at six frequencies ranging linearly from $f = 10$ to $f = 100$ cycles/time. The pressure at node set `INPUT` (nodes at the boundary) is prescribed to have an in-phase component of 3.0 and an out-of-phase component of –4.0 (i.e., a complex value of $3.0 - 4.0i$). An in-phase inward volume acceleration of 40.0 is specified at node 10.

On the surface `LINER1` an impedance is defined based on the impedance property named `CARPET1`. On the second face of all of the elements in element set `PAD`, another surface impedance based on `CARPET1` is defined. On the fourth face of all of the elements in element set `END`, the default plane wave boundary condition is specified.

Printed output of pressure magnitude and phase is requested for node set `OUTPUT`. Acoustic pressure and displacement are written to the output database. All output is written once for each of the six excitation frequencies.

```
*HEADING
…
*SURFACE, NAME=LINER1
 10, S3
*IMPEDANCE PROPERTY, NAME=CARPET1
Data describing impedance properties as a function of frequency
**
*STEP
```

```
*STEADY STATE DYNAMICS, DIRECT
 10, 100, 6
*SIMPEDANCE, PROPERTY=CARPET1
 LINER1,
**
*IMPEDANCE, PROPERTY=CARPET1
 PAD, I2
 *IMPEDANCE
 END, I4
** Apply complex pressure at node set INPUT
*BOUNDARY, REAL
 INPUT, 8, 8, 3.
*BOUNDARY, IMAGINARY
 INPUT, 8, 8, -4.
** Apply an in-phase inward volume acceleration at node 10
*CLOAD
 10, 8, 40.
** Output requests
*NODE PRINT, NSET=OUTPUT, TOTALS=YES
 POR, PPOR
*OUTPUT, FIELD
*NODE OUTPUT
U, PU, POR
*END STEP
```

The following is a template of the step definition for an Abaqus/Explicit acoustic analysis. On the surface `SURF` an impedance is defined based on the impedance property named `IPROP`. In addition, impedance is defined on elements or element sets.

```
*HEADING
…
*ELEMENT, TYPE=AC2D4R
…
**
*SURFACE, NAME=SURF
Data line to define surface
*IMPEDANCE PROPERTY, NAME=IPROP
Data describing impedance properties
**
*STEP
*DYNAMIC, EXPLICIT or *DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
Data line to define incrementation
*SIMPEDANCE, PROPERTY=IPROP
 SURF,
**
*IMPEDANCE
Data lines to define impedance on elements or element sets
*CLOAD
Data line to define acoustic loads
*FIELD
Data line to define field variable values
*END STEP
```

The following template is representative of a coupled acoustic-structural shock problem using the preferred interface for applying incident wave loading (see *Incident Wave Loading due to External Sources*):

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
```

```
Data lines to define solid elements
*ELEMENT, TYPE=…, ELSET=BEAM
Data lines to define beam elements
*BEAM SECTION,ELSET=BEAM,MATERIAL=...
Data lines to define the beam stiffness section properties
*BEAM FLUID INERTIA
Data line to define the beam virtual mass property
*SURFACE, NAME=IW_LOAD_ACOUSTIC
Data lines to define the acoustic surface loaded by the incident wave
*SURFACE, NAME=IW_LOAD_SOLID
Data lines to define the solid surface loaded by the incident wave
*SURFACE, NAME=IW_LOAD_BEAM
Data lines to define the beam surface loaded by the incident wave
*SURFACE, NAME=TIE_ACOUSTIC
Data lines to define the acoustic surface interface with the solid mesh
*SURFACE, NAME=TIE_SOLID
Data lines to define the solid surface interface with the acoustic mesh
*INCIDENT WAVE INTERACTION PROPERTY, NAME=IWPROP, TYPE=SPHERE
Data lines to define a spherical incident wave field
*UNDEX CHARGE PROPERTY
Data lines to define the underwater explosion parameters
** Tie the acoustic mesh to the solid mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
*STEP
*DYNAMIC, EXPLICIT or *DYNAMIC
** Load the acoustic surface
*INCIDENT WAVE INTERACTION, PROPERTY=IWPROP
IW_LOAD_ACOUSTIC, source node, standoff node, reference magnitude
** Load the solid surface
*INCIDENT WAVE INTERACTION, PROPERTY=IWPROP
IW_LOAD_SOLID, source node, standoff node, reference magnitude
** Load the beam surface
*INCIDENT WAVE INTERACTION, PROPERTY=IWPROP
IW_LOAD_BEAM, source node, standoff node, reference magnitude
*END STEP
```

The following template is representative of a coupled acoustic-structural shock problem using the alternative interface for applying incident wave loading:

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define solid elements
*ELEMENT, TYPE=…, ELSET=BEAM
Data lines to define beam elements
*BEAM SECTION,ELSET=BEAM,MATERIAL=...
Data lines to define the beam stiffness section properties
*BEAM FLUID INERTIA
Data line to define the beam virtual mass property
*SURFACE, NAME=IW_LOAD_ACOUSTIC
Data lines to define the acoustic surface loaded by the incident wave
*SURFACE, NAME=IW_LOAD_SOLID
Data lines to define the solid surface loaded by the incident wave
*SURFACE, NAME=IW_LOAD_BEAM
Data lines to define the beam surface loaded by the incident wave
*SURFACE, NAME=TIE_ACOUSTIC
```

```
Data lines to define the acoustic surface interface with the solid mesh
*SURFACE, NAME=TIE_SOLID
Data lines to define the solid surface interface with the acoustic mesh
*INCIDENT WAVE PROPERTY, NAME=IWPROP, TYPE=SPHERE
Data lines to define a spherical incident wave field
*INCIDENT WAVE FLUID PROPERTY
Data lines to define the fluid properties for the incident wave field
*AMPLITUDE, DEFINITION=BUBBLE, NAME=PRESSUREVTIME
Data lines to define the underwater explosion parameters
** Tie the acoustic mesh to the solid mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
*STEP
*DYNAMIC or *DYNAMIC, EXPLICIT
** Load the acoustic surface
*INCIDENT WAVE, PRESSURE AMPLITUDE=PRESSUREVTIME,
PROPERTY=IWPROP
IW_LOAD_ACOUSTIC, {amplitude}
** Load the solid surface and the beam surface
*INCIDENT WAVE, PRESSURE AMPLITUDE=PRESSUREVTIME,
PROPERTY=IWPROP
IW_LOAD_SOLID, {amplitude}
IW_LOAD_BEAM, {amplitude}
*END STEP
```

The following template is representative of a coupled acoustic-structural sound transmission problem using the preferred interface for applying incident wave loading (see *Incident Wave Loading due to External Sources*):

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define solid elements
*SURFACE, NAME=IW_LOAD_ACOUSTIC
Data lines to define the acoustic surface loaded by the incident wave
*SURFACE, NAME=IW_LOAD_SOLID
Data lines to define the solid surface loaded by the incident wave
*SURFACE, NAME=TIE_ACOUSTIC
Data lines to define the acoustic surface interface with the solid mesh
*SURFACE, NAME=TIE_SOLID
Data lines to define the solid surface interface with the acoustic mesh
*INCIDENT WAVE INTERACTION PROPERTY, NAME=FIRST, TYPE=SPHERE
Data lines to define a spherical incident wave field
*INCIDENT WAVE INTERACTION PROPERTY, NAME=SECOND, TYPE=PLANE
Data lines to define a planar incident wave field
** Tie the acoustic mesh to the solid mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
*STEP
*STEADY STATE DYNAMICS, DIRECT or SUBSPACE PROJECTION
** Define the load on the acoustic and solid surfaces due to
** the first loading case:
*LOAD CASE, NAME=FIRST_SOURCE
** Load the acoustic surface: define the real part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=FIRST, REAL
IW_LOAD_ACOUSTIC, first source node, first standoff node, reference magnitude
** Load the acoustic surface: define the imaginary part at the
** standoff point
```

```
*INCIDENT WAVE INTERACTION, PROPERTY=FIRST, IMAGINARY
IW_LOAD_ACOUSTIC, first source node, first standoff node, reference magnitude
** Load the solid surface: define the real part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=FIRST, REAL
IW_LOAD_SOLID, first source node, first standoff node, reference magnitude
** Load the solid surface: define the imaginary part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=FIRST, IMAGINARY
IW_LOAD_SOLID, first source node, first standoff node, reference magnitude
*END LOAD CASE
** Define the load on the acoustic and solid surfaces due to
** the next loading case:
*LOAD CASE, NAME=SECOND_SOURCE
** Load the acoustic surface: define the real part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, REAL
IW_LOAD_ACOUSTIC, second source node, second standoff node, reference magnitude
** Load the acoustic surface: define the imaginary part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, IMAGINARY
IW_LOAD_ACOUSTIC, second source node, second standoff node, reference magnitude
** Load the solid surface: define the real part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, REAL
IW_LOAD_SOLID, second source node, second standoff node, reference magnitude
** Load the solid surface: define the imaginary part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, IMAGINARY
IW_LOAD_SOLID, second source node, second standoff node, reference magnitude
*END LOAD CASE
*END STEP
```

Abaqus/Aqua Analysis

Products: Abaqus/Aqua

Overview

An Abaqus/Aqua analysis:

- is used to apply steady current, wave, and wind loading to submerged or partially submerged structures in problems such as the modeling of offshore piping installations or the analysis of marine risers;
- can be performed using the static (*Static Stress Analysis*), direct-integration dynamic (*Implicit Dynamic Analysis Using Direct Integration*), explicit dynamics (*Explicit Dynamic Analysis*), or eigenfrequency extraction (*Natural Frequency Extraction*) procedures;
- will calculate drag, buoyancy, and inertia loading only for beam, pipe, elbow, truss, and certain rigid elements;
- can include elements that model spud cans for jack-up foundation analysis in Abaqus/Standard; and
- can be linear or nonlinear.

References:

- *UWAVE*
- *Defining an Analysis*
- **AQUA*
- **CLOAD*
- **C ADDED MASS*
- **DLOAD*
- **D ADDED MASS*
- **SURFACE SECTION*
- **WAVE*
- **WIND*

Procedures Available for Aqua Analysis

Aqua loading can be applied in static steps (*Static Stress Analysis*), direct-integration dynamic steps (*Implicit Dynamic Analysis Using Direct Integration*), and explicit dynamic steps (*Explicit Dynamic Analysis*). During these steps fluid particle velocity is assumed to consist of two superposed effects: steady currents, which can vary with elevation and location, and gravity waves. Fluid particle accelerations are associated with gravity waves only.

The fluid particle velocities and accelerations are used to calculate drag and inertia loading on the immersed body. Abaqus/Aqua also computes the fluid surface elevation and allows for partial immersion; drag and buoyancy loadings are omitted for those parts of the structure that are above the fluid surface or below the seabed level.

An eigenfrequency extraction step (*Natural Frequency Extraction*) can be used to extract the natural frequencies of a structure prestressed by the Aqua loading in a static or direct-integration dynamic step (if that step included the effects of nonlinear geometry). The added-mass effect due to fluid inertia loads can be included in an eigenfrequency extraction step.

Defining an Abaqus/Aqua Problem

Aqua loads are applied in the following manner:

1. The fluid properties and steady current velocity are defined for the model.
2. Gravity waves and wind velocity are defined for the model.
3. Drag, buoyancy, and fluid inertia loads are applied to elements and nodes of the structure using distributed or concentrated load definitions within the static or direct-integration dynamic step definition. The magnitudes of the loads applied are determined by the fluid properties, steady current, wave, and wind definitions.
4. In an eigenfrequency extraction step concentrated and distributed added mass definitions are used (instead of concentrated and distributed loads) to include the effects of fluid inertia.

The load-stiffness terms from Abaqus/Aqua loads, which are important in geometrically nonlinear analysis, are fundamentally unsymmetric. Therefore, the unsymmetric matrix solution and storage scheme should be used for the step when nonlinear geometric effects are included (*Defining an Analysis*). It is essential to use the unsymmetric solver when the structure being analyzed is flexible (see, for example, *Slender pipe subject to drag: the "reed in the wind"*).

On the other hand, if a relatively stiff structure is subject to Aqua loads or if a dynamic step uses small time increments, the unsymmetric load-stiffness terms may not be dominant and you may be able to obtain a convergent solution with the symmetric solver (see, for example, *Riser dynamics*).

Coordinate System

The z-coordinate axis must point vertically for three-dimensional cases, and the y-coordinate axis must point vertically for two-dimensional cases. For the three-dimensional case the still fluid surface (when there is no wave motion) lies in a plane that is parallel to the x–y plane. For the two-dimensional case it lies parallel to the x-axis. The position of the still fluid surface is specified as part of the fluid property data.

Defining the Fluid Properties

Aqua loadings require the definition of fluid density, seabed and free surface elevation, and the gravitational constant.

Input File Usage: **AQUA*
seabed elevation, free surface elevation, gravitational constant, fluid density

The **AQUA* option must be included in the model data portion of the input file.

Defining a Steady Current

Steady currents are defined by giving steady fluid velocity as a function of elevation and location. Elevation is defined in the positive z-direction for three-dimensional models and in the positive y-direction for two-dimensional models. For two-dimensional cases the z-component of the steady current velocity is ignored. See *Input Syntax Rules* for an explanation of how to define one property (in this case steady current velocity) as a function of multiple independent variables.

If the fluid velocity is not a function of elevation or location (for example, when modeling a problem in a coordinate system that moves uniformly through the still fluid, such as a tow-out analysis), only one fluid velocity need be specified.

The steady current velocities can be scaled by referring to an amplitude curve (*Amplitude Curves*) from the concentrated or distributed load definitions used to apply drag loads, as described later.

Input File Usage: **AQUA*
fluid properties on first data line (described above)
X-velocity$_{fluid}$, Y-velocity$_{fluid}$, Z-velocity$_{fluid}$, elevation, X-coord, Y-coord
...

Defining Gravity Waves

Gravity waves are defined by specifying a wave theory. The wave theory determines fluid acceleration, velocity, and pressure field fluctuations. The fluid acceleration and velocity field fluctuations contribute to the drag loads. The fluid pressure field fluctuations contribute to the buoyancy loads.

Choosing the Type of Wave Theory to Be Used

Using Abaqus/Aqua in an Abaqus/Standard analysis, you can choose Airy linear wave theory, Stokes fifth-order wave theory, wave data read from a gridded mesh, or fluid kinematics defined in user subroutine *UWAVE*. For Airy and Stokes waves the fluid surface elevation and the fluid particle velocities and accelerations will be calculated as functions of time and location based on the wave definition. If wave data are provided in the form of a gridded mesh, you must specify these quantities. If user subroutine *UWAVE* is used, the fluid kinematics must be defined in that routine.

Similarly, using Abaqus/Aqua in an Abaqus/Explicit analysis, you can choose Airy linear wave theory, Stokes fifth-order wave theory, or fluid kinematics defined in user subroutine *VWAVE*.

All of the built-in wave theories assume a series of waves in the horizontal plane (the plane of the fluid surface) that are unaffected by any fluid-structural interaction. The Airy and Stokes theories are based on irrotational flow of an inviscid, incompressible fluid, where the wave height H is small compared to the still water depth d. The bottom of the fluid is assumed to be flat (the still water depth is constant).

The Ursell parameter,

$$\frac{H}{\lambda}\left(\frac{\lambda}{d}\right)^3,$$

where λ is the wavelength, should be much less than 1.0 for Airy wave theory to be applicable and should be less than 10.0 for Stokes theory to be applicable. For ratios of H/λ greater than 0.142, the crest of the wave is predicted to break. The assumed boundary conditions on the free surface are then no longer valid in either theory, which limits the maximum wave amplitude for either theory.

Airy Wave Theory

Linear Airy wave theory is generally used when the ratio of wave height to water depth, H/d, is less than 0.03, provided that the water is deep (ratio of water depth to wavelength, d/λ, is greater than 20). Convective acceleration terms are neglected in the Airy theory as part of the linearization. The Airy wave theory is described in detail in *Airy wave theory*.

Since the Airy wave theory is linear, any number of wave trains traveling in different directions across the water can be defined; the fluid particle velocities and accelerations sum by linear superposition. The direction of each wave component is given by specifying the direction cosines of a vector, d_N, lying in the plane defined by the still fluid surface.

By default, Airy waves are defined in terms of wavelength, λ_N. Alternatively, you can define the waves in terms of wave period, τ_N. For Airy wave theory the wavelength and period of each component are related by

$$\frac{2\pi}{\tau_N^2} = g\frac{1}{\lambda_N}\tanh\frac{2\pi h}{\lambda_N} \quad ,$$

where

τ_N

is the period of this component,

g

is the gravitational acceleration,

λ_N

is the wavelength, and

h

is the undisturbed (still) water depth.

Input File Usage: Use the following option to define an Airy wave in terms of wavelength:

**WAVE*, TYPE=AIRY
amplitude, wavelength, phase angle, x-direction cosine, y-direction cosine

Use the following option to define an Airy wave in terms of wave period:

**WAVE*, TYPE=AIRY, WAVE PERIOD
amplitude, wave period, phase angle, x-direction cosine, y-direction cosine

In either case repeat the data line to define multiple wave trains.

Stokes Fifth-Order Wave Theory

The Stokes fifth-order wave theory is a deep-water wave theory that is valid for relatively large wavelengths. Convective terms are included in the fluid particle acceleration calculations for Stokes fifth-order theory and can be significant for larger H/λ ratios. The Stokes wave theory is described in detail in *Stokes wave theory*.

Because the Stokes fifth-order wave theory is nonlinear, only one wave train is allowed in an analysis. The relationship between wavelength and period of the waves in Stokes fifth-order theory is not as simple as that for the Airy theory, although the formula given above is a first-order approximation. Stokes waves can be defined only in terms of the wave period, τ.

Input File Usage: **WAVE*, TYPE=STOKES
wave height, wave period, phase angle, direction of travel cosines

Gridded Wave Data

You can choose to provide wave surface elevations, particle velocities and accelerations, and the dynamic pressure at points in a user-defined grid through a binary data file. The binary file contains information about the wave definition, the location of the grid points where wave information is specified, and the wave kinematics at user-defined times. At spatial locations within the user-defined grid, Abaqus/Aqua will interpolate the wave kinematics from the nearest grid points, using either linear or quadratic interpolation. When a point on the structure is above the user-defined grid, Abaqus/Aqua assumes that the point is above the free surface elevation. Hence, no fluid loads are applied. If a point on the structure falls outside the user-defined spatial grid without

being above the grid, Abaqus/Aqua finds the wave kinematics at the nearest point within the grid and uses those values at the point on the structure.

Input File Usage: **WAVE*, TYPE=GRIDDED, DATA FILE=*file_name*

Binary Data File Requirements for Gridded Wave Data

The data file must contain the following unformatted (binary) records (see *Aqua load cases*). The data for the FORTRAN WRITE statement are given for each record:

First record:

```
NCOMP, DTG, NWGX, NWGY, NWGZ, IPDYN
```

where

NCOMP

is the number of wave components to be read in the data file;

DTG

is the time increment at which wave data are given on the grid;

NWGX

is the number of grid points in the grid's *x*-direction;

NWGY

is the number of grid points in the grid's *y*-direction—if this number is one, Abaqus/Aqua assumes that the wave data are constant with respect to the local *y*-direction;

NWGZ

is the number of grid points in the grid's *z*-direction—if this number is zero or one, the analysis is two-dimensional and the *y*-direction is vertical; and

IPDYN

is an integer flag indicating whether dynamic pressure information is stored (`IPDYN`=1) or not stored (`IPDYN`=0) in the gridded wave file.

Second record:

```
(AMP(K1), WXL(K1), PHI(K1), K1=1,NCOMP)
```

where

NCOMP

is read on the first record, above;

AMP

contains the wave component amplitude, a_N;

WXL

contains the wavelength of this component, λ_N; and

PHI

contains the phase angle of this component, ϕ_N (in degrees).

The second record of this file contains the wave component data used to generate the gridded wave data; it is not used by Abaqus/Aqua. This record is provided only for information in user subroutine *UEL* by using the GETWAVE interface (see *Obtaining Wave Kinematic Data in an Abaqus/Aqua Analysis*). The meaning of the arrays AMP and WXL is left to you; however, PHI is converted to radians.

Third record:

```
(WGX(K1),K1=1,NWGX), (WGY(K1),K1=1,NWGY), (WGZ(K1),K1=1,NWGZ)
```

where

NWGi

are read on the first record, above;

WGX

contains the local *x*-coordinates of the grid points;

WGY

contains the local *y*-coordinates of the grid points; and

WGZ

contains the local *z*-coordinates of the grid points (not included in the gridded wave file for two-dimensional analyses).

Remaining records if IPDYN=0:

For three dimensions:

```
(((WGVX(K1,K2,K3), WGVY(K1,K2,K3), WGVZ(K1,K2,K3),
WGAX(K1,K2,K3), WGAY(K1,K2,K3), WGAZ(K1,K2,K3), K3=1,NWGZ),
WZCRST(K1,K2), NCRST(K1,K2), K1=1,NWGX), K2=1,NWGY)
```

For two dimensions:

```
((WGVX(K1,K2), WGVY(K1,K2), WGAX(K1,K2), WGAY(K1,K2),
K2=1,NWGY), WZCRST(K1), NCRST(K1), K1=1,NWGX)
```

Remaining records if IPDYN=1:

For three dimensions:

```
(((WGVX(K1,K2,K3), WGVY(K1,K2,K3), WGVZ(K1,K2,K3),
WGAX(K1,K2,K3), WGAY(K1,K2,K3), WGAZ(K1,K2,K3),
P(K1,K2,K3), DPDZ(K1,K2,K3), K3=1,NWGZ),
WZCRST(K1,K2), NCRST(K1,K2), K1=1,NWGX), K2=1,NWGY)
```

For two dimensions:

```
((WGVX(K1,K2), WGVY(K1,K2), WGAX(K1,K2), WGAY(K1,K2),
P(K1,K2), DPDZ(K1,K2), K2=1,NWGY),
WZCRST(K1), NCRST(K1), K1=1,NWGX)
```

where

WGVX

contains the local *x*-components of the wave particle velocity,

WGVY

contains the local *y*-components of the wave particle velocity,

WGVZ

contains the local *z*-components of the wave particle velocity,

WGAX

contains the local *x*-components of the wave particle acceleration,

WGAY

contains the local *y*-components of the wave particle acceleration,

WGAZ

contains the local *z*-components of the wave particle acceleration,

WZCRST

contains the wave surface elevation,

NCRST

contains the index for the vertical grid level just above the instantaneous water surface,

P

contains the dynamic pressure, and

DPDZ

contains the gradient of the dynamic pressure in the vertical direction.

User-Defined Wave Theory in Abaqus/Standard

A user-defined wave theory can be coded in user subroutine *UWAVE* in an Abaqus/Aqua analysis in Abaqus/Standard. You can define the fluid particle velocity, acceleration, free surface elevation, and fluid pressure field in the user subroutine.

For stochastic analysis, you can specify a random number seed, *r*, and define frequency/amplitude pairs that define the wave spectrum. During the analysis Abaqus/Aqua stores an intermediate configuration that can be used in the user subroutine to compute the stochastic description of the waves. The intermediate configuration is initialized as the reference configuration and is replaced by the current configuration only when requested by the user subroutine. In this way the stochastic description of the wave field can be stored in an external database and recalculated only when necessary.

Input File Usage: Use the following option to specify the wave kinematics in user subroutine *UWAVE*:

**WAVE*, TYPE=USER

Use the following option for stochastic analysis to make the intermediate configuration available in user subroutine *UWAVE*:

**WAVE*, TYPE=USER, STOCHASTIC=*r*
frequency, amplitude
...

User-Defined Wave Theory in Abaqus/Explicit

A user-defined wave theory can be coded in user subroutine *VWAVE* in an Abaqus/Aqua analysis in Abaqus/Explicit. You can define the fluid particle velocity, acceleration, free surface elevation, and fluid pressure field in the user subroutine.

The quantities required to define the wave kinematics can be specified as properties and passed into the user subroutine. For example, in the case of stochastic wave kinematics, any required seed variable and/or frequency-amplitude data pairs can be specified as properties.

You can also declare and use state variables for user-defined wave calculations, which will be provided at the nodes and initialized to zero at the beginning of the step. You have to update the state variables within the user subroutine. For example, the state variables can be used to store any intermediate configuration of the structure that is used to describe a stochastic wave field.

Input File Usage: Use the following option to specify the wave kinematics in user subroutine *VWAVE*:

**WAVE*, TYPE=USER

Use the following option to specify properties available as a real-array argument `PROPS` of size `NPROPS` in user subroutine *VWAVE*:

**WAVE*, TYPE=USER, PROPERTIES=*nprops*
prop_1, prop_2, ..., prop_8
..., prop_nprops

Use the following option to specify state variables available as a real-array argument `STATEVAR` of size `NSTATEVAR` in user subroutine *VWAVE*:

**WAVE*, TYPE=USER, DEPVAR=*nstatevar*

Wave Position as a Function of Time

For Airy and Stokes waves the position of the wave at time $t = 0$ can be chosen by specifying the phase angle ϕ of the wave (or wave components for Airy waves). By default, the waves are chosen such that they have a trough (vertical displacement of the fluid surface is a minimum) at the origin of the horizontal axes at time $t = 0$.

You can change this trough by introducing a phase angle ϕ for the waves. A positive phase angle shifts the waves backward in their travel direction (see *Figure 1*).

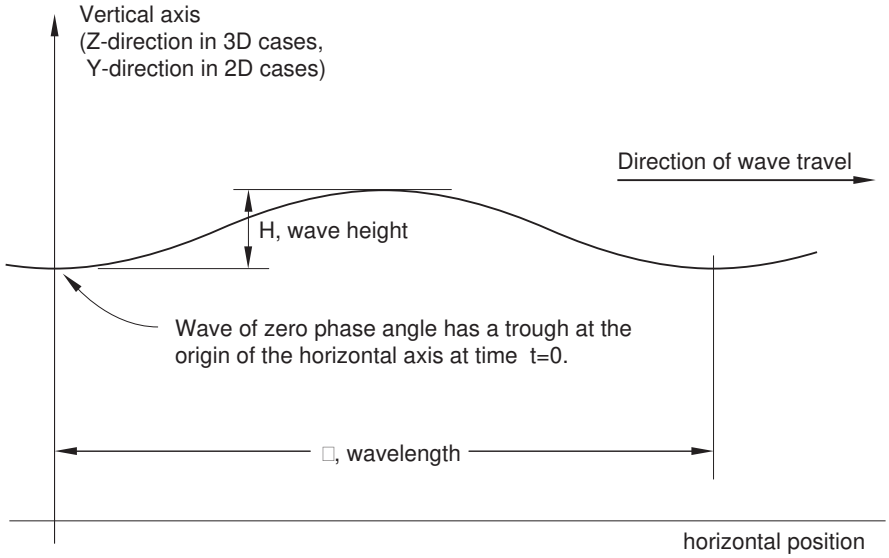


Figure 1: Wave of zero phase angle.

The time *t* used in the wave theory is the total time in the analysis. Therefore, if the direct-integration dynamic steps in which Airy or Stokes waves are applied are preceded by any steps other than direct-integration dynamic steps (such as static steps), it is usually convenient to make the time period in these steps very small compared to the period of the wave.

Because total time is used, the phase of the wave will be continuous from the end of one dynamic step to the beginning of the next dynamic step.

Defining a Minimum Wave Trough Elevation

For computational efficiency Abaqus/Aqua uses a minimum wave trough elevation below which the structure is assumed to be immersed. Below this elevation no calculation of the fluid surface need be done to determine if the point of interest is above the instantaneous free surface. Similarly, a maximum wave elevation is used: any point above the maximum wave elevation is assumed to have no fluid loading.

For Airy and Stokes waves the minimum and maximum wave elevations are calculated from the wave theory.

For gridded waves Abaqus/Aqua allows the definition of a minimum wave trough elevation: z_{min} in three-dimensional analysis or y_{min} in two-dimensional analysis. The structure is always assumed to be immersed below this elevation. The maximum wave elevation is calculated as the still water elevation plus the difference between this elevation and the minimum wave trough elevation. If the minimum wave trough elevation is not specified for gridded waves, Abaqus/Aqua will compare the elevation of every point on the structure with the instantaneous fluid surface as defined by the gridded data. When defining this elevation, make sure that no wave trough ever drops below the minimum wave trough elevation specified.

Input File Usage: **WAVE*, TYPE=GRIDDED, DATA FILE=*file_name*, MINIMUM=*elevation*

Wave Kinematics, Dynamic Pressure, and Extrapolation for Airy Waves

A spatial (Eulerian) description of the wave field is used for all wave types; therefore, a structural point's coordinates are used to evaluate the wave kinematics. In geometrically nonlinear analysis the structural point's coordinates are its current coordinates. In geometrically linear analysis the wave kinematics are evaluated using the structural point's reference coordinates.

In both geometrically linear and nonlinear analysis for both static and direct-integration dynamic procedures, submergence is calculated to the instantaneous water level at the current value of total time for the analysis. Fluid loading is applied only to those points on the structure below the instantaneous water level.

When buoyancy loading is applied in conjunction with a gravity wave, the dynamic pressure due to the disturbance of the still surface is added to the hydrostatic pressure (measured to the still water level) to obtain the total buoyancy loading, except when the buoyancy loading described by a distributed or concentrated load definition overrides the fluid properties given for the Abaqus/Aqua analysis. Dynamic pressure is included for both static and dynamic procedures for Airy, Stokes, and gridded wave types; however, with gridded wave data you can choose to suppress this effect. See *Airy wave theory* and *Stokes wave theory* for a definition of dynamic pressure.

Although the linearized Airy wave theory assumes that the fluid displacements are small with respect to the wavelength and the fluid depth, these displacements may not be small with respect to the dimensions of the structure immersed in the fluid. As a result of the linearizing approximations special treatment is necessary to calculate the wave kinematics for points below the instantaneous water level but above the still water line. Abaqus/Aqua uses extrapolation with Airy wave theory: the wave velocity, acceleration, and dynamic pressure for points above the still water level but below the instantaneous free surface are taken to be the values evaluated from the wave theory at the still water level. See *Airy wave theory* for more details.

Reading the Data That Define Gravity Waves from an Alternate File

The data for the gravity wave can be contained in an alternate file. See *Input Syntax Rules* for the syntax of the file name.

Input File Usage: **WAVE*, INPUT=*file_name*

Visualization of Gravity Waves

In a three-dimensional analysis you can visualize gravity waves by meshing the free surface of the water with surface elements (see *General Surface Element Library*) and identifying elements as aqua visualization elements through the surface section definition.

Aqua visualization elements are used for postprocessing only and do not affect the solution. The following must be true for proper use of these elements:

1. Aqua visualization elements can be connected to other visualization elements only through shared nodes. They cannot be connected in any way to any element in the model that is used during the analysis. This includes connections through shared nodes, kinematic constraints, or surface interactions. Abaqus issues an error message during input file preprocessing if these conditions are not met. For example, if you are doing an Abaqus/Aqua analysis of an offshore oil platform, the visualization elements cannot be connected to any element used to model the platform.
2. Any boundary conditions or loads that are applied on the visualization elements are ignored.
3. Density cannot be assigned to the visualization elements.
4. Reinforcement layers cannot be defined for the visualization elements.

5. To visualize the displacements, you must request displacement field output on the output database (`.odb`) file. During the analysis Abaqus computes the z-displacements of the elements using whatever wave definitions you include in the model, including user subroutines. Only displacement output can be requested for these elements.
6. The initial z-coordinates of the elements should be defined at the still water height; if they are not, Abaqus automatically adjusts them to the still water height during input file preprocessing.

Input File Usage: **SURFACE SECTION*, ELSET=*elset_name*, AQUAVISUALIZATION=YES

Defining a Wind Velocity Profile

You can define a wind velocity profile. Wind loading is applied only to elements above the still water surface elevation (defined in the fluid properties). If an element is above the still water depth but is submerged due to a wave, the wind loading will still be applied.

The wind profile is assumed to vary with height (the positive z-direction in three-dimensional models, the positive y-direction in two-dimensional models) according to the power law wind profile and has no variation in the horizontal plane. The power law wind velocity profile is given by

$$\mathbf{v} = \mathbf{v}^0 \left(\frac{z}{z_0} \right)^{\alpha},$$

where

$\mathbf{v}(z,t)$

$= v_x(z,t)\,\hat{\mathbf{i}} + v_y(z,t)\,\hat{\mathbf{j}}$ is the local wind velocity ($\hat{\mathbf{i}}$ is a unit vector along the local x-axis of the wind field, and $\hat{\mathbf{j}}$ is a unit vector along the local y-axis of the wind field);

$\mathbf{v}^0(t)$

is the time-varying wind velocity at the reference height, z_0, as described below;

α

is a user-defined constant (default value 1/7);

z

is the distance above the still water surface (i.e., $z = 0.0$ is the still water surface); and

z_0

is the reference distance above the still water surface where the time variation of the wind velocity is given.

The wind local system is defined by giving the direction cosines of the unit vector $\hat{\mathbf{i}}$.

Input File Usage: **WIND*

air density, z_0, c_x, c_y, *x-direction cosine for* $\hat{\mathbf{i}}$, *y-direction cosine for* $\hat{\mathbf{i}}$, α

Prescribing the Time Variation of Wind Velocity at the Reference Height

The variation in time of the wind profile is defined by $\mathbf{v}^0(t)$, the wind velocity vector time history at a reference height $z = z_0$:

$$\mathbf{v}^0(t) = v_x^0(t)\,\hat{\mathbf{i}} + v_y^0(t)\,\hat{\mathbf{j}}.$$

The wind velocity component time histories $v_x^0(t)$ and $v_y^0(t)$ are given by

$$v_x^0(t) = c_x A_x(t)\,; \quad v_y^0(t) = c_y A_y(t)\,,$$

where c_x and c_y are user-defined as described above (with default values of 1.0) and $A_x(t)$ and $A_y(t)$ are time-dependent functions defined by referring to amplitude curves from the concentrated or distributed load definitions used to apply the wind loading to the model. If no amplitude curve is referenced, the wind velocity components are the constant values $v_x^0(t) = c_x$ and $v_y^0(t) = c_y$.

Geometrically Linear Versus Geometrically Nonlinear Analysis

In geometrically linear analysis wind velocities are calculated based on the original coordinates of the structure. In geometrically nonlinear analysis the current coordinates of a point on the structure are used to calculate the wind velocity at that point.

Initial Conditions

Initial conditions can be applied to the structure in an Abaqus/Aqua analysis in the same way as in static and dynamic analyses without Aqua loads. See *Initial Conditions*.

Boundary Conditions

Boundary conditions can be applied to the structure in an Abaqus/Aqua analysis in the same way as in static and dynamic analyses without Aqua loads. See *Boundary Conditions*.

Defining Contact at the Seabed

Aqua loads are applied only above the seabed. To model the bottom of the sea using a contact plane, the elevation of the contact plane must be slightly higher than the seabed level to avoid ambiguity between the contact condition and applied loading. If the contact plane is at the same level as the seabed, there is a risk that round-off problems will cause Aqua loads not to be applied to nodes in contact with the seabed.

Loads

Steady current, wave, and wind loads are applied to nodes or elements of the structure using concentrated and/or distributed load definitions. Wind loads are applied only if the point is currently above the still fluid surface; fluid loads are applied only if the point is currently below the instantaneous fluid surface and above the seabed. Distributed loads are applied to partly immersed elements.

Concentrated and distributed load definitions cannot be used in eigenfrequency extraction steps, so the loads described below can be applied only in static and direct-integration dynamic steps.

Controlling the Time Variation and Magnitude of Aqua Loading

You have three ways to control the magnitude of an Aqua load as a function of time:

1. You can reference a user-defined amplitude curve (*Amplitude Curves*) from the concentrated or distributed load definition to scale the entire load.
2. You can specify a magnitude factor, M, for the concentrated or distributed load definition, which is used to scale all the load. This magnitude factor allows normalized amplitude curves to be defined and used for multiple loads. The default magnitude factor is always $\boldsymbol{M = 1.0}$.
3. You can reference individual user-defined amplitude curves to scale different components of the loading separately. For example, steady current velocity and wave velocity can be scaled separately by referencing different amplitude curves.

All of these scaling factors are cumulative.

Buoyancy Loads

The calculated buoyancy of a structure depends on the orientation of the exposed surface area with respect to the vertical direction. This surface area is calculated automatically by Abaqus/Aqua for distributed buoyancy loading; however, you must specify the exposed area and direction cosines of the outward normal at a node for concentrated buoyancy loading.

Abaqus/Aqua uses a closed-end loading condition while computing the distributed buoyancy forces on all line elements. To obtain an open-end loading condition, concentrated buoyancy loading can be used to counteract the buoyancy load applied to the ends of the elements.

The buoyancy loads require the definition of fluid density, seabed and free surface elevation, and the gravitational constant. The default external fluid properties are defined for the model as described in *Defining the Fluid Properties*. You can override some of these properties by specifying them directly in the distributed or concentrated load definition. This provides for modeling situations where different parts of the structure are subjected to different buoyancy loads, such as a pipe inside another pipe where the static fluid surrounding the inner pipe is different from the fluid surrounding the outer pipe. Gravity waves (*Wave Kinematics, Dynamic Pressure, and Extrapolation for Airy Waves*) do not affect the buoyancy loading when any external fluid property is overridden.

Specifying Distributed Buoyancy Loads

To apply distributed buoyancy loads to elements immersed in a fluid, the effective outer diameter of beam, truss, and one-dimensional rigid elements must be specified. Provide the external fluid density, free surface elevation, and additional pressure to override the default fluid properties to model the situations described above. For situations where it is necessary to model the fluid inside an element, the effective inner diameter of the element must also be given, along with the density and free surface elevation of the fluid inside the element.

Distributed buoyancy loading can be applied to rigid surface elements. However, the effects of waves are ignored for these elements; the buoyancy loading is calculated to the still water level only. For proper application of a positive buoyancy force, the positive normal of R3D3 and R3D4 elements must point into the fluid.

Input File Usage: **DLOAD*
element number or set, PB, *M*, *effective outer diameter*, *internal fluid density, effective inner diameter, internal free surface elevation, external fluid density, external free surface elevation, additional pressure*

Specifying Concentrated Buoyancy Loads

For concentrated buoyancy loads applied to nodes immersed in a fluid, the load is calculated based on the sum of the hydrostatic pressure (measured to the still water level) and the dynamic pressure due to wave action. The total pressure is multiplied by the exposed area associated with the node. The loading is automatically considered to be a follower force in geometrically nonlinear analysis (for elements that have rotational degrees of freedom); therefore, it is not necessary to specify that the load is a follower force. Provide the external fluid density, free surface elevation, and additional pressure to override the default fluid properties to model the situations described above.

Input File Usage: **CLOAD*
node number or set, TSB, *M*, *exposed area*, *local coordinate system data, external fluid density, external free surface elevation, additional pressure*

Drag Loads

Both waves and wind can cause drag loading on a structure. Fluid drag refers to drag caused by the structural member being immersed in the fluid defined by the fluid properties and the gravity waves and, thus, subject to steady current and wave loading. Fluid drag loading is provided by Morison's equation. Fluid drag loads must be specified in terms of a normal (transverse) load and a tangential load.

Wind drag is generated on the portions of a structure that are above the still fluid surface defined by the fluid properties because these portions are exposed to the user-defined wind velocity profile.

Specifying Distributed Transverse Fluid or Wind Drag Loads

Distributed transverse drag is defined as follows (see *Drag, inertia, and buoyancy loading* for more details):

$$\mathbf{F}_D = A\tfrac{1}{2}\rho C_D D\Delta\mathbf{v}_{fn}\left|\Delta\mathbf{v}_{fn}\right|,$$

where

$\mathbf{F}_D$

is the force per unit length, transverse to the member;

$A(t)$

is the current value of the amplitude curve referred to by the distributed load definition, multiplied by the user-defined magnitude factor, *M*;

ρ

is the mass density of the fluid (given in the fluid properties) for fluid distributed drag or is the mass density of the air (given in the wind velocity profile) for wind distributed drag;

C_D

is the drag coefficient; and

D

is the effective outer diameter of the member.

The relative fluid particle velocity in the normal direction, $\Delta\mathbf{v}_{fn}$, is given by

$$\begin{aligned}\Delta\mathbf{v}_{fn} &= \Delta\mathbf{v} - \Delta\mathbf{v}\cdot\mathbf{t}\,\mathbf{t},\\ \Delta\mathbf{v} &= \mathbf{v}_f - \alpha_R\mathbf{v}_p,\end{aligned}$$

where

$\mathbf{v}_f$

is the fluid particle velocity (see the discussion below);

$\mathbf{v}_p$

is the velocity of this point on the structure (zero during static steps);

α_R

is the structural velocity factor; and

$\mathbf{t}$

is the unit vector along the axis of the element.

The effective outer diameter of the element, D; the drag coefficient, C_D; and the structural velocity factor, α_R, must be defined in the distributed load definition together with the distributed load type (fluid distributed drag or wind distributed drag).

The velocities due to steady current and waves can be scaled individually for fluid distributed drag by referring to different amplitude curves. Thus, the fluid particle velocity, $\mathbf{v}_f$, at any time is

$$\mathbf{v}_f = A_c\mathbf{v}_c + A_w\mathbf{v}_w,$$

where

$A_c\,(t)$

is the current value of the first amplitude curve listed in the load definition or 1.0 if the amplitude reference is omitted,

$\mathbf{v}_c$

is the steady current velocity defined in the fluid properties,

$A_w\,(t)$

is the current value of the second amplitude curve listed in the load definition or 1.0 if the amplitude reference is omitted, and

$\mathbf{v}_w$

is the user-defined wave velocity.

The wind velocity is defined in components relative to the local axes $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ defined for the wind velocity profile. Each velocity component can be scaled independently by referring to different amplitude curves. The total wind velocity at any time, $\mathbf{v}_f$, is

$$\mathbf{v}_f = \left(c_x A_x \hat{\mathbf{i}} + c_y A_y \hat{\mathbf{j}}\right) \left(\frac{z}{z_0}\right)^{\alpha},$$

where $A_x\,(t)$ and $A_y\,(t)$ are the amplitude references provided in the load definition for the velocity components in the local x- and y-directions, respectively. The values of c_x, c_y, z_0, and α are defined by the wind velocity profile; and z is the distance above the still fluid surface.

Input File Usage: Use the following option to define fluid distributed drag:

**DLOAD*

element number or set, FDD, *M*, *D*, C_D, α_R, $A_c\,(t)$, $A_w\,(t)$

Use the following option to define wind distributed drag:

**DLOAD*

element number or set, WDD, *M*, *D*, C_D, α_R, $A_x\,(t)$, $A_y\,(t)$

Specifying Distributed Tangential Fluid Drag Loads

Distributed tangential fluid loading is a load in the tangential direction of an element due to skin friction. This type of loading is defined as follows (see *Drag, inertia, and buoyancy loading* for more details):

$$\mathbf{F}_t = A\tfrac{1}{2}\rho_w C_t \pi D \Delta\mathbf{v}_{ft} |\Delta\mathbf{v}_{ft}|^{h-1},$$

where

$\mathbf{F}_t$

is the force per unit length, tangent to the member;

$A\,(t)$

is the amplitude curve referred to by the distributed load definition, multiplied by the user-defined magnitude factor, M;

ρ_w

is the mass density of the fluid (given in the fluid properties);

C_t

is the tangential drag coefficient;

D

is the effective outer diameter of the member; and

h

is a constant (by default, $h = 2$, for quadratic dependence of force on velocity).

The relative fluid particle velocity in the tangential direction, $\Delta\mathbf{v}_{ft}$, is given by

$$\Delta\mathbf{v}_{ft} = (\mathbf{v}_f - \alpha_R\mathbf{v}_p) \cdot \mathbf{tt},$$

where

$\mathbf{v}_f$

is the fluid particle velocity (as defined above for distributed transverse fluid drag loading),

$\mathbf{v}_p$

is the velocity of this point on the structure (zero during static steps),

α_R

is the structural velocity factor, and

$\mathbf{t}$

is the unit vector along the axis of the element.

The effective outer diameter of the element, D; the drag coefficient, C_t; the structural velocity factor, α_R; and the exponent, h, must be defined in the distributed load definition together with the distributed load type (fluid drag tangential).

As with distributed transverse fluid loading, the velocities due to steady current and waves (A_c and A_w) can be scaled individually by referring to different amplitude curves.

Input File Usage: Use the following option to define fluid drag tangential:

**DLOAD*

element number or set, FDT, M, D, C_t, α_R, h, $A_c(t)$, $A_w(t)$

Specifying Concentrated Fluid or Wind Drag Loads Using a Concentrated Load Definition

Concentrated fluid or wind drag loading applies a load normal to the end of an element. Such loading is automatically considered to be a follower force in geometrically nonlinear analysis (for elements that have rotational degrees of freedom).

The drag theory uses Morison's equation (see *Drag, inertia, and buoyancy loading*). The drag force is nonzero when the net flow is in the opposite direction of the outward normal to the exposed area and is zero when the net flow is in the direction of the normal:

$$\mathbf{F}_{\text{drag}} = \begin{cases} -A\frac{1}{2}\rho C_n \Delta A (\Delta v_{ft})^2 \mathbf{t} & \text{for} \quad \Delta v_{ft} \le 0, \\ 0 & \text{for} \quad \Delta v_{ft} > 0, \end{cases}$$

where

$A(t)$

is the amplitude curve referenced by the concentrated load definition multiplied by the user-defined magnitude factor, *M*;

ρ

is the mass density of the fluid (given in the fluid properties) for transition section fluid drag or is the mass density of the air (given in the wind velocity profile) for transition section wind drag;

C_n

is the drag coefficient;

ΔA

is the exposed area; and

Δv_{ft}

is the relative velocity between the structural member and the fluid particle along $\mathbf{t}$ and is given by $\mathbf{t} \cdot \Delta \mathbf{v}_{ft}$, where $\Delta \mathbf{v}_{ft} = (\mathbf{v}_f - \alpha_R \mathbf{v}_p) \cdot \mathbf{tt}$, as defined above for distributed tangential fluid drag loading.

The exposed area, ΔA; the drag coefficient, C_n; and the structural velocity factor, α_R, must be defined in the concentrated load definition together with the concentrated load type (transition section fluid drag or transition section wind drag).

As with distributed transverse fluid loading, the velocities due to steady current and waves (A_c and A_w) and the velocity components of the wind in the $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ directions (A_x and A_y) can be scaled individually by referring to different amplitude curves.

Input File Usage: Use the following option to define transition section fluid drag:

**CLOAD*

node number or set, TFD, *M*, ΔA, C_n, α_R, $A_c(t)$, $A_w(t)$

Use the following option to define transition section wind drag:

**CLOAD*

node number or set, TWD, *M*, ΔA, C_n, α_R, $A_x(t)$, $A_y(t)$

Specifying Concentrated Fluid or Wind Drag Loads Using a Distributed Load Definition

You can apply concentrated fluid or wind drag loading on the ends of elements. These loads have the same effect as specifying a concentrated load at a node using a concentrated load definition with concentrated load type transition section fluid drag or transition section wind drag, except that the normal to the exposed area cannot be specified when a distributed load definition is used; the normal to the end of the element is defined by the tangent to the element.

The load can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element. These loads are nonzero only when the net flow is in the opposite direction of the outward normal to the exposed area.

The loading is exactly the same as that described for the concentrated fluid or wind drag loading applied with a concentrated load definition. The "distributed" form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid drag on the first end of the element:

**DLOAD*

element number or set, FD1, M, ΔA, C, α_R, $A_c\,(t)$, $A_w\,(t)$

Use the following option to define fluid drag on the second end of the element:

**DLOAD*

element number or set, FD2, M, ΔA, C, α_R, $A_c\,(t)$, $A_w\,(t)$

Use the following option to define wind drag on the first end of the element:

**DLOAD*

element number or set, WD1, M, ΔA, C, α_R, $A_x\,(t)$, $A_y\,(t)$

Use the following option to define wind drag on the second end of the element:

**DLOAD*

element number or set, WD2, M, ΔA, C, α_R, $A_x\,(t)$, $A_y\,(t)$

Neglecting the Wave's Contribution to Drag and Inertia Loading during a Step

If the wave's contribution to the drag and inertia loading should not be applied during a step, the concentrated or distributed load component definition must explicitly refer to an amplitude curve with a value of zero. This is the only way to prevent waves from contributing to the fluid velocities and accelerations used in the calculation of these concentrated or distributed load types.

Fluid Inertia Loads (Added-Mass Effects)

Fluid inertia loading causes a structure to have increased inertial resistance to acceleration. This fluid "added-mass" effect is included automatically in a direct-integration dynamic step when fluid inertia loading is applied. Concentrated or distributed added mass must be defined to include the added-mass effect in an eigenfrequency extraction step.

Specifying Distributed Fluid Inertia Loads in a Direct-Integration Dynamic Step

Distributed fluid inertia loading is defined as follows (see *Drag, inertia, and buoyancy loading* for a more detailed description):

$$\mathbf{F}_I = A\rho_w \tfrac{\pi D^2}{4} \left[C_M\, \mathbf{a}_{fn} - C_A\, \mathbf{a}_{pn}\right],$$

where

$\mathbf{F}_I$

is the force per unit length, transverse to the member, caused by fluid inertia;

$A(t)$

is the amplitude curve referred to by the distributed load definition multiplied by the user-defined magnitude factor, M;

ρ_w

is the mass density of the fluid (given in the fluid properties);

D

is the effective outer diameter of the member;

C_M

is the transverse fluid inertia coefficient;

C_A

is the transverse added-mass coefficient;

$\mathbf{a}_{fn}$

is the transverse component of the fluid acceleration; and

$\mathbf{a}_{pn}$

is the transverse component of the beam acceleration (zero during static steps).

The effective outer diameter, D; transverse fluid inertia coefficient, C_M; and transverse added-mass coefficient, C_A, must be defined in the distributed load definition together with the distributed load type (distributed fluid inertia).

The fluid acceleration, $\mathbf{a}_{fn}$, is calculated according to the user-defined gravity wave and is further scaled by the amplitude curve, A_w, referred to by the distributed load definition.

Input File Usage: Use the following option to define distributed fluid inertia in a dynamic step:

**DLOAD*

element number or set, FI, M, D, C_M, C_A, $A_w(t)$

Specifying Distributed Fluid Inertia Loads in an Eigenfrequency Extraction Step

The added mass contribution due to distributed fluid inertia loading is

$$\rho_w \frac{\pi D^2}{4} C_A$$

per unit length of the member in the directions transverse to the axis of the member only, where

ρ_w

is the mass density of the fluid (given in the fluid properties),

D

is the effective outer diameter of the member, and

C_A

is the transverse added-mass coefficient.

Input File Usage: **D ADDED MASS*
element number or set, FI, *D*, C_A

Specifying Concentrated Fluid Inertia Loads in a Direct-Integration Dynamic Step Using a Concentrated Load Definition

Concentrated fluid inertia loading is automatically considered to be a follower force (for elements that have rotational degrees of freedom).

The inertia term is calculated as a force in the current direction of the outward normal to the exposed surface area:

$$\mathbf{F}_I = A\rho_w \left(K_{ts} F_{1s} \mathbf{a}_{ft} - L_{ts} F_{2s} \mathbf{a}_{pt}\right),$$

where

$\mathbf{F}_I$

is the point force caused by fluid inertia;

$A(t)$

is the amplitude curve referenced by the concentrated load definition multiplied by the user-defined magnitude factor, *M*;

ρ_w

is the mass density of the fluid (given in the fluid properties);

K_{ts}

is the tangential inertia coefficient;

F_{1s}

is the fluid acceleration shape factor (of dimension L^3);

L_{ts}

is the tangential added-mass coefficient;

F_{2s}

is the structural acceleration shape factor (of dimension L^3);

$\mathbf{a}_{ft}$

is the fluid acceleration in the direction of the outward normal to the exposed surface; and

$\mathbf{a}_{pt}$

is the structural acceleration in the direction of the outward normal to the exposed surface (zero during static steps).

The tangential inertia coefficient, K_{ts}; the fluid acceleration shape factor, F_{1s}; the tangential added-mass coefficient, L_{ts}; and the structural acceleration shape factor, F_{2s}, are given in the concentrated load definition together with the concentrated load type (transition section inertia).

The fluid acceleration, $\mathbf{a}_{ft}$, is calculated according to the user-defined gravity wave and is further scaled by the amplitude curve, A_w, referred to by the concentrated load definition.

Input File Usage: Use the following option to define transition section inertia in a dynamic step:

**CLOAD*

node number or set, TSI, *M*, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w\,(t)$

Specifying Concentrated Fluid Inertia Loads in a Direct-Integration Dynamic Step Using a Distributed Load Definition

You can apply concentrated fluid inertia loading at the ends of elements. These loads have the same effect as specifying a concentrated fluid added-inertia loading using a concentrated load definition with concentrated load type transition section inertia, except that the normal to the exposed area cannot be specified when a distributed load definition is used; the normal to the end of the element is defined by the tangent to the element.

The inertia loading can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element.

The loading is exactly the same as that described for the concentrated fluid inertia loading applied with a concentrated load definition. The “distributed” form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid inertia on the first end of the element in a dynamic step:

**DLOAD*

element number or set, FI1, *M*, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w\,(t)$

Use the following option to define fluid inertia on the second end of the element in a dynamic step:

**DLOAD*

element number or set, FI2, *M*, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w\,(t)$

Specifying Concentrated Fluid Inertia Effects in an Eigenfrequency Extraction Step Using a Concentrated Added Mass Definition

The added mass contribution due to concentrated fluid inertia loading in an eigenfrequency extraction step is

$$\rho_w L_{ts} F_{2s}$$

in the direction normal to the transition section area, where

ρ_w

is the mass density of the fluid (given in the fluid properties),

L_{ts}

is the tangential added-mass coefficient, and

F_{2s}

is the structural acceleration shape factor (of dimension L^3).

Input File Usage: **C ADDED MASS*

node number or set, TSI, L_{ts}, F_{2s}
direction cosines defining the outward normal of the exposed area

Specifying Concentrated Fluid Inertia Effects in an Eigenfrequency Extraction Step Using a Distributed Added Mass Definition

You can apply concentrated fluid inertia effects at the ends of elements. These loads have the same effect as specifying concentrated fluid inertia effects using a concentrated added mass definition with concentrated load type transition section inertia, but in this case the normal to the exposed area cannot be specified; the normal to the end of the element is defined by the tangent to the element.

The added mass can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element.

The effect is exactly the same as that described for the concentrated fluid inertia effects applied with a concentrated added mass definition. The “distributed” form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid inertia on the first end of the element in an eigenfrequency extraction step:

**D ADDED MASS*

element number or set, FI1, L_{ts}, F_{2s}

Use the following option to define fluid inertia on the second end of the element in an eigenfrequency extraction step:

**D ADDED MASS*

element number or set, FI2, L_{ts}, F_{2s}

Applying Non-Aqua Loads to the Structure

Concentrated and distributed load definitions can also be used to apply concentrated and distributed forces that are not associated with wind, waves, or steady current to the structure. See *Concentrated Loads* and *Distributed Loads*.

Predefined Fields

The following predefined fields can be specified for the structure (not the fluid) in an Abaqus/Aqua analysis, as described in *Predefined Fields*:

- Temperatures of nodes in the structure can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material Options

Any of the mechanical constitutive models in Abaqus can be used for modeling the structure in an Abaqus/Aqua analysis (see *Abaqus Materials Guide* for details on the material models available in Abaqus/Standard).

Elements

The fluid loads in an Abaqus/Aqua analysis cannot be applied to all element types. Only the beam, pipe, elbow, truss, and rigid beam elements in Abaqus/Standard and linear beam and pipe elements in Abaqus/Explicit can be used to subject a structure to general Abaqus/Aqua loading. The only load that can be applied to two-dimensional rigid surfaces (R3D3 and R3D4 elements) is hydrostatic buoyancy; and this loading can be applied only in Abaqus/Standard. Current, wave, and wind loading have no effect on rigid surfaces.

Jack-Up Foundation Analysis

Abaqus/Standard provides element types JOINT2D and JOINT3D, which can be used to model elastic-plastic interaction between spud cans and the sea floor (see *Elastic-Plastic Joints*).

Output

In addition to the usual output variables available in Abaqus/Standard (see *Abaqus/Standard Output Variable Identifiers*) and in Abaqus/Explicit (see *Abaqus/Explicit Output Variable Identifiers*), element section output variable ESF1 can be used to request output of the effective axial force in a beam subjected to pressure loading (see *Beam Element Library*). The velocities and accelerations of the fluid cannot be output.

Input File Template

```
*HEADING
…
*SURFACE SECTION,ELSET=aquaviz,AQUAVISUALIZATION=YES
*NSET,NSET=naquaviz,ELSET=aquaviz
*AQUA
Data lines defining the fluid properties and steady current velocity
*WAVE, TYPE=wave theory
Data lines defining gravity waves
**
*STEP (, NLGEOM)
Use the NLGEOM parameter to include nonlinear geometric effects
*DYNAMIC (or *STATIC or *DYNAMIC, EXPLICIT)
…
*CLOAD
Data lines defining concentrated buoyancy, fluid/wind drag, and fluid inertia loads
```

```
*DLOAD
Data lines defining distributed buoyancy, fluid/wind drag, and fluid inertia loads
*OUTPUT, FIELD, TIME INTERVAL=interval for field output
*NODE OUTPUT,NSET=naquaviz
U
*END STEP
**
*STEP
The NLGEOM parameter must have been included in the previous step to obtain
the natural frequencies of the prestressed structure
*FREQUENCY
…
*C ADDED MASS
Data lines to define concentrated added-mass effects
*D ADDED MASS
Data lines to define distributed added-mass effects
*OUTPUT, FIELD, TIME INTERVAL=interval for field output
*NODE OUTPUT,NSET=naquaviz
U
*END STEP
```

Annealing

Products: Abaqus/Explicit Abaqus/CAE

Overview

The anneal procedure:

- is used to anneal a structure by setting all appropriate state variables and velocities to zero;
- is intended for metal plasticity and user-defined material models; and
- can also be used with linear elastic materials and with superelastic materials for modeling shape setting in Abaqus/Explicit.

It has no effect on other material models.

References:

- **ANNEAL*
- *Configuring an annealing procedure*

The Annealing Process

The anneal procedure is intended to simulate the relaxation of stresses and plastic strains that occurs as metals are heated to high temperatures. Physically, annealing is the process of heating a metal part to a high temperature to allow the microstructure to recrystallize, removing dislocations caused by cold working of the material. During the anneal procedure, Abaqus/Explicit sets all appropriate state variables to zero. These variables include stresses, backstresses, plastic strains, and velocities. In the case of metal porous plasticity, the void volume fraction is also set to zero, such that the material becomes fully dense.

There is no time scale in an annealing step; therefore, time does not advance. The annealing process occurs instantaneously. No data are required for the anneal procedure.

Input File Usage:

**ANNEAL*

Abaqus/CAE Usage:

Step module: **Create Step**: **General**: **Anneal**

Temperatures

Thermal strains are set to zero, and the temperature at all nodes in the model will be set to a uniform temperature or will be maintained at the current temperature during the anneal procedure. By default, the temperature at all nodes is maintained at the current temperature. You can specify a different final temperature, θ.

Input File Usage:

**ANNEAL*, TEMPERATURE=θ

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Anneal**: **Post-anneal reference temperature: Value** θ

Initial Conditions

The initial state for the anneal step is the state of the model at the end of the last explicit dynamic analysis step.

Boundary Conditions

It is not appropriate to specify new boundary conditions or to modify boundary conditions in an anneal procedure; all boundary conditions in effect prior to this procedure will remain fixed.

Loads

It is not meaningful to specify loads in an anneal procedure.

Predefined Fields

It is not meaningful to specify predefined fields in an anneal procedure.

Material Options

The annealing procedure is intended for metal plasticity models (*Classical Metal Plasticity*) and user-defined materials modeled with user subroutines *VFABRIC* and *VUMAT*. The metal plasticity models in Abaqus/Explicit include Mises, Johnson-Cook, Hill, and metal porous plasticity. Abaqus/Explicit also allows annealing of linear elastic materials (*Linear Elastic Behavior*, including isotropic, orthotropic, and anisotropic elasticity), as well as annealing of superelastic materials for modeling shape setting (*Superelasticity*). The annealing procedure has no effect on other material models.

Elements

All of the elements that are available in Abaqus/Explicit can be used in an anneal procedure. The elements are listed in *Abaqus Elements Guide*.

Output

There is no output associated with an anneal step.

Input File Template

```
*HEADING
 …
**
*STEP
*DYNAMIC, EXPLICIT (,ADIABATIC) or
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
Data line to specify the time period of the step
*BOUNDARY, AMPLITUDE=name
Data lines to describe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
```

```
*END STEP
**
*STEP
*ANNEAL (,TEMPERATURE=θ)
*END STEP
**
*STEP
*DYNAMIC, EXPLICIT (,ADIABATIC)
Data line to specify the time period of the step
*BOUNDARY, AMPLITUDE=name
Data lines to describe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*END STEP
```

One-Step Inverse Analysis

Products: Abaqus/Standard

References:

- **ONE-STEP INVERSE*

Overview

One-step inverse analysis in Abaqus/Standard:

- is used to obtain the initial shape of a sheet metal part given its final (deformed) configuration and a reference blank surface to which the initial configuration must conform;
- neglects the history of the forming process by assuming proportional loading (Hencky's deformation theory of plasticity);
- provides an accurate method for trim line development in flanging operations by unfolding the flange onto the reference blank surface;
- provides an estimate of plastic deformation induced during forming without detailed modeling of the forming process;
- ignores the effects of frictional forces between the tools and the part;
- is available only with shell elements;
- is a static procedure that neglects inertia effects;
- uses unsymmetric matrix storage and solution; and
- is always geometrically nonlinear.

Introduction

The one-step inverse method (*Batoz et al.*) has become a key analysis tool during the early stages of digital planning of sheet metal stamping processes. The method requires only prior knowledge of the final part design, the material properties, and the initial blank surface and thickness. Given this information, an inverse finite element method is used to compute the initial shape of the part on the blank surface (usually a flat surface). The method is particularly useful in the early stages of part design. You can use the method to, among other things, assess the formability of the part and estimate the blank size when tooling geometry is not yet available.

Another important application of the method is trim line development in sheet metal flanging operations. The one-step inverse method can be used to unfold the flange from its final (as designed) configuration onto an initial blank surface. The trim line can be identified from the perimeter of the unfolded flange. The current capabilities available in Abaqus/Standard are particularly suitable for flange unfolding applications as well as other forming operations with relatively simple boundary conditions, such as clamping a sheet edge.

One-step inverse analysis requires the following input:

- Shell mesh of the final (deformed) configuration.
- Shell section definition, including initial (undeformed) sheet thickness and material definition.
- Mesh of the blank surface that acts as the constraint for the solution of the undeformed configuration.

- Initial estimate of the undeformed configuration (usually based on geometric projection) to act as the initial guess for the nonlinear Newton method.
- Node set representing the region of the model with prescribed boundary conditions based on the forming process; the boundary conditions should also be sufficient to prevent rigid body modes.

The one-step inverse method does not consider the intermediate history of the forming process. Instead, it assumes that a proportional loading path is followed between the initial and final configuration. In addition, the implementation of the method in Abaqus/Standard ignores the effect of friction forces on the part. Therefore, the accuracy of the results depends on the validity of these assumptions for a given forming process. The results of the analysis include the nodal coordinates in the initial configuration (as defined by an inverse displacement vector) and the material state (stress, plastic strain, thickness, etc.) in the final configuration. The method is supported only with shell elements.

Part Definition

The definition of the part requires that you specify a mesh of shell elements that conforms to the part geometry in its final configuration. In addition to the usual element connectivity and nodal coordinates, it is recommended that the nodal definition also include information about the nodal surface normals based on the geometry of the part. These normals are needed to evaluate curvature changes with respect to the undeformed configuration. If the nodal normals are not defined directly, Abaqus computes them by averaging the normals of the elements connected to the node.

Input File Usage:

**NODE*
data lines to specify nodal coordinates and (optional but recommended) normals in the final configuration

Incrementation

One-step inverse analysis is always performed in one increment. If a material model includes plastic behavior, the analysis is performed first without taking plasticity into account. Next, this solution is used as a starting point in the analysis with the complete material model.

Input File Usage:

**ONE-STEP INVERSE*, UNFOLD

Controlling the Solution Accuracy

The one-step inverse method is an iterative procedure that requires a stopping criterion—a condition that, when satisfied, considers the solution of the current iteration to be the final converged solution. During the iterations of a one-step inverse analysis, Abaqus monitors the maximum incremental correction to the undeformed nodal coordinates. When this value becomes smaller than a correction tolerance, the solution is considered to have converged. The correction tolerance is determined as the product of the reference shell section thickness and a thickness ratio. The default value of the thickness ratio is 10^{-2} and can be modified using the field solution controls.

The default for the maximum number of equilibrium iterations is 16. You can modify this value using the time incrementation solution controls.

At every equilibrium iteration Abaqus performs line search iterations if the total norm of the residual vector is not decreased by a prescribed factor. The default value of the maximum number of line search iterations is 3, and the default value of the residual reduction factor is 1.0. You can modify these default values using the line search solution controls.

Input File Usage: Use the following option to modify the value of the thickness ratio:

**CONTROLS*, PARAMETERS=FIELD
, thickness_ratio

Use the following option to modify the value for the maximum number of equilibrium iterations:

**CONTROLS*, PARAMETERS=TIME INCREMENTATION
, , , maximum_number_of_equilibrium_iterations

Use the following option to modify the value for the maximum number of line search iterations and the residual reduction factor:

**CONTROLS*, PARAMETERS=LINE SEARCH
maximum_number_of_line_search_iterations, , , residual_reduction_factor

Initial Conditions

One-step inverse analysis uses an iterative Newton method to determine the undeformed configuration of the part given its final configuration. The undeformed configuration must conform to a surface (the blank surface) that acts as a constraint to the solution. The Newton method requires an initial estimate of the solution as a starting point in the iterative algorithm. You must define this initial configuration by specifying initial conditions for the unfolded coordinates of all nodes in the part (see *Defining the Initial Configuration in a One-Step Inverse Analysis*). The algorithm may fail to converge if this initial configuration is not sufficiently close to the final solution or if it violates the surface constraint significantly. Therefore, a good initial guess of the undeformed (or unfolded) configuration can be critical to the success of the method. A number of CAD packages offer tools to generate an initial guess based on geometric projections of a mesh over a surface.

In addition, initial values of temperatures, field variables, solution-dependent state variables, etc. can be specified, as described in *Initial Conditions*.

Input File Usage:

**INITIAL CONDITIONS*, TYPE=UNFOLD COORDINATE
data lines to specify initial conditions of undeformed nodal coordinates

Boundary Conditions

To make the problem well posed, some of the nodes of the structure must be constrained. You can use fixed boundary conditions to indicate the constrained nodes (see *Fixing Degrees of Freedom at a Point in an Abaqus/Standard Analysis*). All of the degrees of freedom of the specified nodes are constrained based on their initial undeformed configuration, which was specified by initial conditions.

In the case of flange unfolding applications, there is usually a folding line that represents the intersection of the folded flange geometry and the reference blank surface. The nodes on the folding line should be constrained with a fixed boundary condition.

For other applications, such as deep drawing of a metal cup, you should define fixed boundary conditions for a set of nodes (for example, along the perimeter of the part), such that their displacements are fully prescribed from the difference between the nodal coordinates in the final configuration and the undeformed configuration (defined by the initial conditions). More complex boundary conditions that represent blank holding forces or draw beads are not supported.

Input File Usage: **BOUNDARY*, FIXED
data lines to describe fixed boundary conditions

Loads

No loads can be prescribed in a one-step inverse analysis.

Predefined Fields

The following predefined fields can be specified in a one-step inverse analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in a one-step inverse analysis, you can specify nodal temperatures. Any difference between the applied and initial temperatures causes thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- You can specify values of user-defined field variables. These values affect only field variable–dependent material properties, if any.

Material Options

In one-step inverse analysis, isotropic elastic and elastic-plastic material models are supported (with transverse anisotropy is allowed; see *Defining Transversely Isotropic Elasticity*). Using other material models may lead to convergence difficulties.

Elements

The following shell element types are supported in a one-step inverse analysis to model the part: S3/S3R, S4, S4R, and STRI3 elements. You can use any element type to model the blank.

Constraints

You must define the blank surface on which you expect to position the (unknown) undeformed configuration of the part (see *About Surfaces*). The blank surface can be any type of surface that is allowed as a main surface in Abaqus/Standard contact calculations. The blank surface acts as a main surface in a no-separation contact pair definition, while the secondary surface must include all the elements of the part to be analyzed (see *Using the No Separation Relationship*). During the one-step inverse solution process, the no-separation contact constraint is applied only to the reference configuration of the part, thereby enforcing the solution of the undeformed configuration to be on this surface.

Input File Usage: **CONTACT PAIR*, INTERACTION=*interaction_property_name*
secondary_surface_name, main_surface_name
**SURFACE INTERACTION*, NAME=*interaction_property_name*
**SURFACE BEHAVIOR*, NO SEPARATION

Output

In addition to the usual output variables available in Abaqus/Standard (see *Abaqus/Standard Output Variable Identifiers*), the following nodal variable is provided specifically for one-step inverse procedures:

UINV Inverse displacement. The displacement from the specified final deformed configuration to the initial configuration.

Displacement U is always zero in a one-step inverse analysis.

You can visualize the initial configuration of the part in Abaqus/Viewer by creating a deformed plot using UINV as the deformed variable.

Limitations

A one-step inverse analysis is subject to the following limitations:

- Only one step is allowed in a one-step inverse analysis.
- General shell section behavior cannot be used.
- The model of the part must consist of connected regions.

Input File Template

```
*HEADING
.....
*NODE
Data lines to specify nodal coordinates and (optional) normals in the final configuration
*INITIAL CONDITIONS, TYPE=UNFOLD COORDINATE
Data lines to specify initial undeformed nodal coordinates
*CONTACT PAIR, INTERACTION=interaction_property_name
secondary_surface_name, main_surface_name
*SURFACE INTERACTION, NAME=interaction_property_name
*SURFACE BEHAVIOR, NO SEPARATION
**
*STEP (,NLGEOM)
*ONE-STEP INVERSE, UNFOLD
*BOUNDARY, FIXED
Data lines to describe fixed boundary conditions
*END STEP
```

References

- Batoz, J. L., Y. Q. Guo, and F. Mercier, “The Inverse Approach with Simple Triangular Shell Elements for Large Strain Predictions of Sheet Metal Forming Parts,” *Engineering Computations*, vol. 15, pp. 864–892, 1998.

Analysis Solution and Control

In this section:

- *Solving Nonlinear Problems*
- *Analysis Convergence Controls*

Solving Nonlinear Problems

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Convergence and Time Integration Criteria*
- *Commonly Used Control Parameters*
- *Convergence Criteria for Nonlinear Problems*
- *Time Integration Accuracy in Transient Problems*
- *Configuring general analysis procedures*

Overview

Solving nonlinear problems in Abaqus/Standard involves:

- a combination of incremental and iterative procedures;
- using the Newton method to solve the nonlinear equations;
- determining convergence;
- defining loads as a function of time; and
- choosing suitable time increments automatically.

Alternative techniques to the standard Newton method are available. Some static problems may become unstable because of severe nonlinearity. Abaqus/Standard offers a set of automatic stabilization mechanisms to handle such problems.

The Solution of Nonlinear Problems

The nonlinear load-displacement curve for a structure is shown in *Figure 1*.

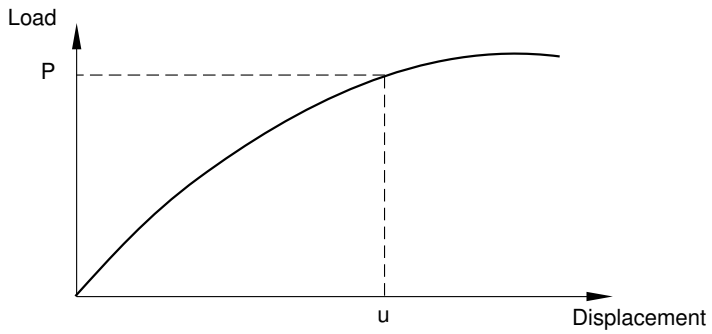


Figure 1: Nonlinear load-displacement curve.

The objective of the analysis is to determine this response. In a nonlinear analysis the solution cannot be calculated by solving a single system of linear equations, as would be done in a linear problem. Instead, the solution is found by specifying the loading as a function of time and incrementing time to obtain the nonlinear response. Therefore, Abaqus/Standard breaks the simulation into a number of *time increments* and finds the approximate

equilibrium configuration at the end of each time increment. Using the Newton method, it often takes Abaqus/Standard several iterations to determine an acceptable solution to each time increment.

Steps, Increments, and Iterations

- The time history for a simulation consists of one or more *steps*. You define the steps, which generally consist of an analysis procedure, loading, and output requests. Different loads, boundary conditions, analysis procedures, and output requests can be used in each step. For example:

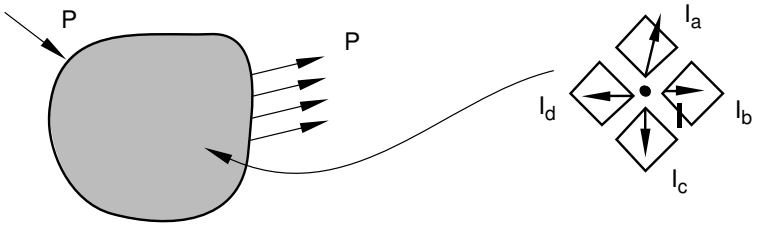
  Step 1: Hold a plate between rigid jaws.

  Step 2: Add loads to deform the plate.

  Step 3: Find the natural frequencies of the deformed plate.
- An *increment* is part of a step. In nonlinear analyses each step is broken into increments so that the nonlinear solution path can be followed. You suggest the size of the first increment, and Abaqus/Standard automatically chooses the size of the subsequent increments. At the end of each increment the structure is in (approximate) equilibrium and results are available for writing to the restart, data, results, or output database files.
- An *iteration* is an attempt at finding an equilibrium solution in an increment. If the model is not in equilibrium at the end of the iteration, Abaqus/Standard tries another iteration. With every iteration the solution that Abaqus/Standard obtains should be closer to equilibrium; however, sometimes the iteration process may diverge—subsequent iterations may move away from the equilibrium state. In that case Abaqus/Standard may terminate the iteration process and attempt to find a solution with a smaller increment size.

Convergence

Consider the external forces, *P*, and the internal (nodal) forces, *I*, acting on a body (see *Figure 2*(a) and *Figure 2*(b), respectively). The internal loads acting on a node are caused by the stresses in the elements that are attached to that node.



(a) External loads in a simulation. (b) Internal forces acting at a node.

Figure 2: Internal and external loads on a body.

For the body to be in equilibrium, the net force acting at every node must be zero. Therefore, the basic statement of equilibrium is that the internal forces, *I*, and the external forces, *P*, must balance each other:

$$\boldsymbol{P - I = 0.}$$

The nonlinear response of a structure to a small load increment, $\boldsymbol{\Delta P}$, is shown in *Figure 3*. According to the Newton method, Abaqus/Standard uses the structure's tangent stiffness, $\boldsymbol{K_0}$, calculated from its configuration at $\boldsymbol{u_0}$, and $\boldsymbol{\Delta P}$ to calculate a *displacement correction*, $\boldsymbol{c_a}$, for the structure. Using $\boldsymbol{c_a}$, the structure's configuration is updated to $\boldsymbol{u_a}$.

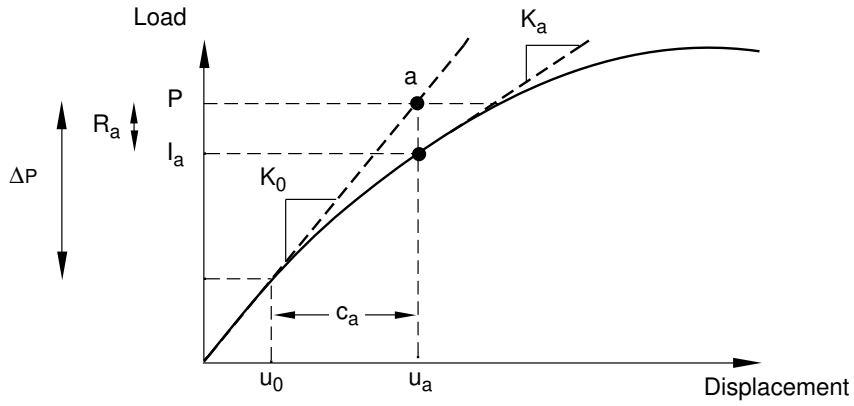


Figure 3: First iteration in an increment.

Abaqus/Standard then calculates the structure's internal forces, $\boldsymbol{I_a}$, in this updated configuration. The difference between the total applied load, P, and $\boldsymbol{I_a}$ can now be calculated as

$$\boldsymbol{R_a = P - I_a},$$

where $\boldsymbol{R_a}$ is the *force residual* for the iteration.

If $\boldsymbol{R_a}$ is zero at every degree of freedom in the model, point *a* in *Figure 3* would lie on the load-deflection curve and the structure would be in equilibrium. In a nonlinear problem $\boldsymbol{R_a}$ will never be exactly zero, so Abaqus/Standard compares it to a tolerance value. If $\boldsymbol{R_a}$ is less than this force residual tolerance at all nodes, Abaqus/Standard accepts the solution as being in equilibrium. By default, this tolerance value is set to 0.5% of an average force in the structure, averaged over time. Abaqus/Standard automatically calculates this spatially and time-averaged force throughout the simulation. You can change this, and all other such tolerances, by specifying solution controls (see *Convergence Criteria for Nonlinear Problems*).

If $\boldsymbol{R_a}$ is less than the current tolerance value, P and $\boldsymbol{I_a}$ are considered to be in equilibrium and $\boldsymbol{u_a}$ is a valid equilibrium configuration for the structure under the applied load. However, before Abaqus/Standard accepts the solution, it also checks that the last displacement correction, $\boldsymbol{c_a}$, is small relative to the total incremental displacement, $\boldsymbol{\Delta u_a = u_a - u_0}$. If $\boldsymbol{c_a}$ is greater than a fraction (1% by default) of the incremental displacement, Abaqus/Standard performs another iteration. Both convergence checks must be satisfied before a solution is said to have *converged* for that time increment.

If the solution from an iteration is not converged, Abaqus/Standard performs another iteration to try to bring the internal and external forces into balance. First, Abaqus/Standard forms the new stiffness, $\boldsymbol{K_a}$, for the structure based on the updated configuration, $\boldsymbol{u_a}$. This stiffness, together with the residual $\boldsymbol{R_a}$, determines another displacement correction, $\boldsymbol{c_b}$, that brings the system closer to equilibrium (point *b* in *Figure 4*).

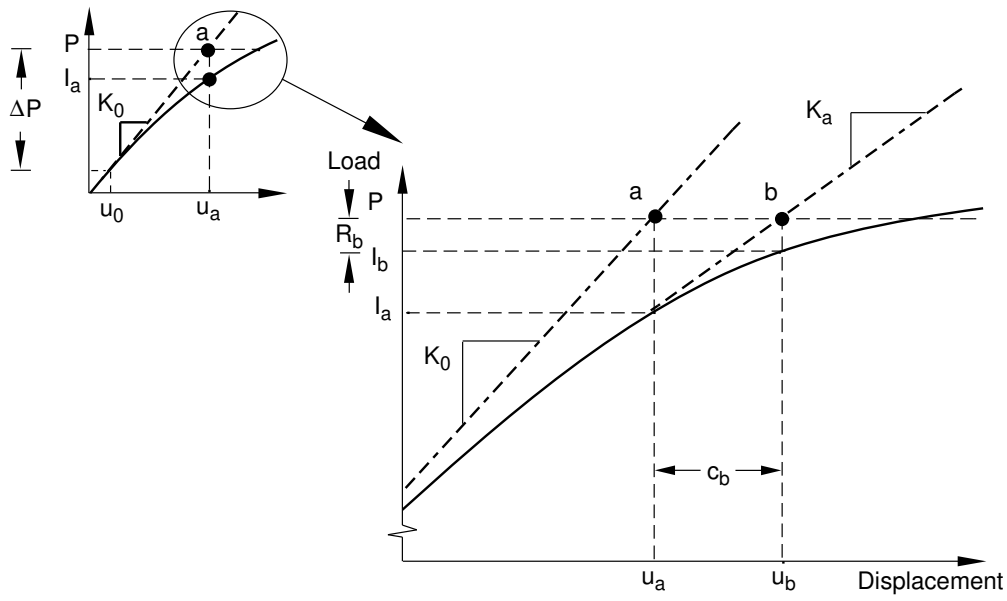


Figure 4: Second iteration.

Abaqus/Standard calculates a new force residual, R_b, using the internal forces from the structure's new configuration, u_b. Again, the largest force residual at any degree of freedom, R_b, is compared against the force residual tolerance, and the displacement correction for the second iteration, c_b, is compared to the increment of displacement, Δu_b. If necessary, Abaqus/Standard performs further iterations. For more details on convergence in Abaqus/Standard, see *Convergence Criteria for Nonlinear Problems*.

For each iteration in a nonlinear analysis Abaqus/Standard forms the model's stiffness matrix and solves a system of equations. Therefore, the computational cost of each iteration is close to the cost of conducting a complete linear analysis, making the computational expense of a nonlinear analysis potentially many times greater than the cost of a linear analysis. Since it is possible with Abaqus/Standard to save results at each converged increment, the amount of output data available from a nonlinear simulation can also be much greater than that available from a linear analysis of the same geometry.

Solution Method

Abaqus/Standard by default uses the Newton's method to solve nonlinear problems iteratively (see section *Convergence* for a description). In some cases it uses an exact implementation of Newton's method, in the sense that the Jacobian or the stiffness matrix of the system is defined exactly, and quadratic convergence is obtained when the estimate of the solution is within the radius of convergence of the algorithm. In other cases the Jacobian is approximated so that the iterative method is not an exact Newton method. For example, some material and surface interface models (such as nonassociated flow plasticity models or Coulomb friction) create a nonsymmetric Jacobian matrix, but you may choose to approximate this matrix by its symmetric part.

Alternative techniques to the standard Newton method are available. These techniques attempt to reduce costs associated with assembly of the full Jacobian and its subsequent factorization during every iteration in the Newton method. The details of these techniques and the situations where they are applicable to derive performance gains are described in the following.

Specifying the Quasi-Newton Method

You can choose to use the quasi-Newton technique for a particular step (described in *Quasi-Newton solution technique*) instead of the standard Newton method for solving nonlinear equations.

The quasi-Newton technique can save substantial computational cost in some cases by reducing the number of times the Jacobian matrix is factorized. Generally it is most successful when the system is large and many iterations are needed per increment or when the stiffness matrix is not changing much from iteration to iteration (such as in a dynamic analysis using implicit time integration or in a small-displacement analysis with local plasticity). It can be used only for symmetric systems of equations; therefore, it cannot be used when the unsymmetric solver is specified for a step (see *Defining an Analysis*), nor can it be used for procedures that always produce an unsymmetric system of equations, such as *Fully Coupled Thermal-Stress Analysis* and *Abaqus/Aqua Analysis*. In addition, it cannot be used for a static Riks procedure (see *Unstable Collapse and Postbuckling Analysis*).

The quasi-Newton method works well in combination with the line search method (see *Improving the Efficiency of the Solution by Using the Line Search Algorithm*). Line searches help to prevent divergence of equilibrium iterations resulting from the inexact Jacobian produced by the quasi-Newton method. The line search method is activated by default for steps that use the quasi-Newton method. You can override this action by specifying line search controls.

You can specify the number of quasi-Newton iterations allowed before the kernel matrix is reformed. The default number of iterations is 8. Additional matrix reformations may occur automatically during the iteration process depending on the convergence behavior. Since quadratic convergence is not expected during quasi-Newton iterations, the logarithmic rate of convergence check is not applied during the time incrementation. Furthermore, the iteration count used in the time incrementation is a weighted sum of quasi-Newton iterations, with the weight factor depending on whether or not a kernel matrix has been reformed.

The quasi-Newton solution technique is not available when the model contains matrices included with the matrix input options.

Input File Usage: **SOLUTION TECHNIQUE*, TYPE=QUASI-NEWTON, REFORM KERNEL=*n*

Abaqus/CAE Usage: Step module: step editor: **Other**: **Solution technique: Quasi-Newton**, **Number of iterations allowed before the kernel matrix is reformed:** *n*

Specifying the Separated Method

Alternatively, you can choose to use the separated technique instead of the standard Newton method for solving nonlinear equations for fully coupled thermal-stress and coupled thermal-electrical procedures.

The separated technique (described in *Fully Coupled Thermal-Stress Analysis* and *Coupled Thermal-Electrical Analysis*) approximates the Jacobian by eliminating interfield coupling terms and can save substantial computational cost in cases where there is relatively weak coupling between the fields.

Input File Usage: **SOLUTION TECHNIQUE*, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: step editor: **Other**: **Solution technique: Separated**

Linear Complementarity Problem (LCP) Solution Technique for Solving Contact Problems

Contact problems in Abaqus/Standard involve solving nonlinear equations in general nonlinear steps. For each load increment, the active set of contact constraints and corresponding contact stresses are solved for along with other solution variables (typically) using Newton iterations. For structural problems involving contact, an alternative solution technique is available under special modeling assumptions listed below.

- Contact can be modeled entirely with the small sliding formulation with frictionless tangential behavior. Small sliding can be specified with contact pairs or a nondefault setting with general contact.

- Symmetric stiffness matrix storage is applicable.
- Except contact, the rest of the model response can be assumed to be linear.

Under these assumptions, the governing equations take the form of a set of linear inequality constraints from contact (contact gap has to be either positive or zero) along with a set of linear equations for equilibrium. The resulting mathematical problem is known as the Linear Complementarity Problem (LCP). The LCP technique solves for unknown active contact constraints and corresponding contact stresses together with displacement unknowns during the solution process. This solution technique is accessible only within a static perturbation step (see *Contact within Static Perturbation Analysis*). The linearization of material and geometric nonlinearities at the base state is relied upon for ensuring linear response for the rest of the model automatically while nonlinearities only due to contact status changes are handled. Commonly, geometric nonlinearities are turned off as a modeling assumption while using the LCP approach.

The LCP solution involves the following phases. First, displacement degrees-of-freedom are condensed out and the governing equations are written only in terms of unknown contact pressure variables from potential contact constraints at the contact interfaces. The potential constraints are determined based on gap/overclosure distances at the base state prior to the LCP solution process. The condensation phase is followed by an iterative process (LCP iterations) to determine active contact constraints and the corresponding contact pressure values. The displacement solution is obtained from the contact pressure solution in a subsequent recovery phase. Global stiffness matrix assembly and solving the global set of equations during multiple nonlinear iterations in the traditional approach are avoided, which makes the LCP approach more efficient in some situations.

Input File Usage: Use the following option to activate the LCP solution technique:

**SOLUTION TECHNIQUE*,TYPE=*LCP CONTACT*

Abaqus/CAE Usage: The LCP solution technique is not supported in Abaqus/CAE.

Specifying a Gap Distance to Control Potential Constraints Exposed to the LCP Solver

The number of potential constraints exposed to the LCP solver can be controlled by a gap distance that you specify. All closed contact constraints and those that are open and within the gap distance in the base state are included as potential constraints. By default, a gap distance is internally computed based on a characteristic facet dimension associated with individual contact pairs. You can also provide a scale factor to scale the gap distance that you specified or the internally computed defaults. The choice of the gap distance affects performance.

Choosing a conservative gap distance can rapidly increase the cost of the LCP solver since such costs show a cubic growth in the number of potential contact constraints. The LCP solver can automatically revise the gap distance while keeping a check on the number of potential constraints if it finds that the initial choice of the gap distance is small and results in overclosure violations. Therefore, it may not be necessary to be overly conservative while choosing the gap distance.

For an example of this technique, see *Input File Template*.

Input File Usage: Use the following option to specify the gap distance that specifies the potential contact zone and a scale factor for the gap distance for the LCP solution technique:

**SOLUTION TECHNIQUE*,TYPE=LCP CONTACT
gap distance, scale factor

Abaqus/CAE Usage: The LCP solution technique is not supported in Abaqus/CAE.

Additional Perspectives

Several examples are presented here for a better understanding of the performance characteristics of the LCP approach.

Figure 5 shows an example where a solid sphere is pinched between rigid plates. Linear elastic material behavior is assigned to the sphere, while the small-sliding formulation without friction is assumed for modeling contact. Geometric nonlinearities are turned off. This model is set up so that both the traditional and the LCP approaches are expected to give the same final solution. Boundary conditions are applied to push the top plate down in a static step for the traditional approach, and the same conditions are prescribed within a static perturbation step in the LCP approach. An initial guess of half the boundary displacement is specified for the gap distance to include potential contact constraints.

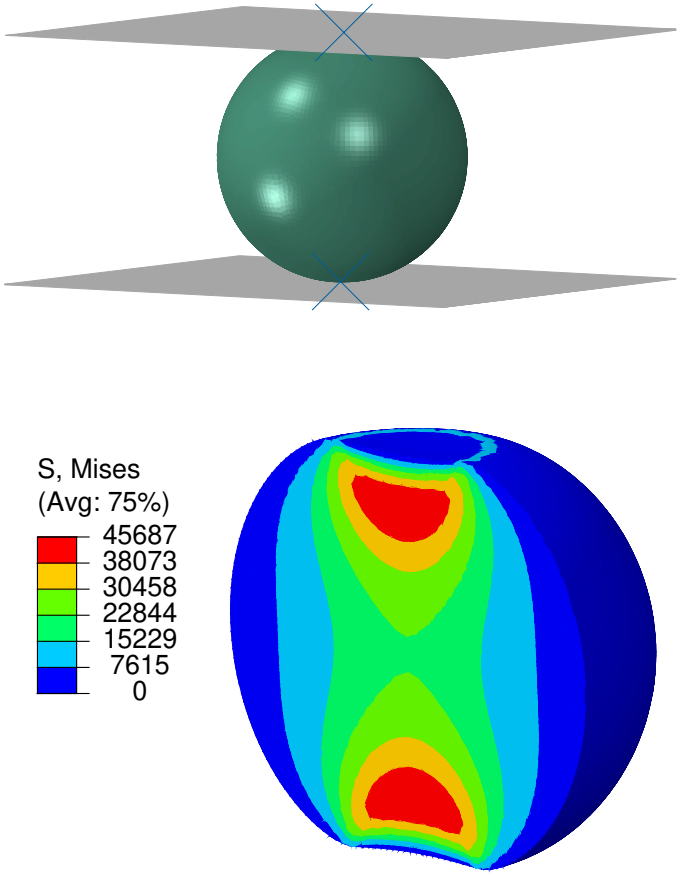


Figure 5: Pinching of a solid sphere solved with LCP contact.

Table 1 compares normalized wall times between the traditional and the LCP approaches for three different meshes for the model in *Figure 5*. The LCP approach shows significant speedups in this example.

Table 1: Performance comparison of LCP contact with traditional contact for a pinched sphere.

Quantity	Coarse mesh		Medium mesh		Fine mesh	
	LCP approach	Traditional approach	LCP approach	Traditional approach	LCP approach	Traditional approach
Normalized wall time (speed up factor for LCP)	1.0	5.0 (5.0)	27.8	193 (6.9)	187	1925 (10.3)
Number of global assembly and solve iterations	1	7	1	9	1	10
Number of LCP iterations	134	Not applicable	492	Not applicable	1044	Not applicable
Model size	42,000		306,000		890,000	
Number of potential contact constraints to LCP solver	353	Not applicable	1496	Not applicable	3476	Not applicable
Active constraints from the solution	187		743		1696	

Consider instead a grid of spheres in *Figure 6* made by assembling instances of the setup in the previous example. Three grids with 10, 50 and 100 spheres are considered for comparison purposes in *Table 2*. These models more closely approximate an assembly of parts. The meshes on the individual spheres correspond to the coarsest mesh in the previous example. The rest of the model definitions remain the same. From *Table 2* it is clear that as the number of potential contact constraints increase, the cubic growth of the cost of the LCP solver starts to dominate. The performance degradation compared to the traditional approach is rapid as the grid size increases and becomes unacceptable for the 50 to the 100 sphere models.

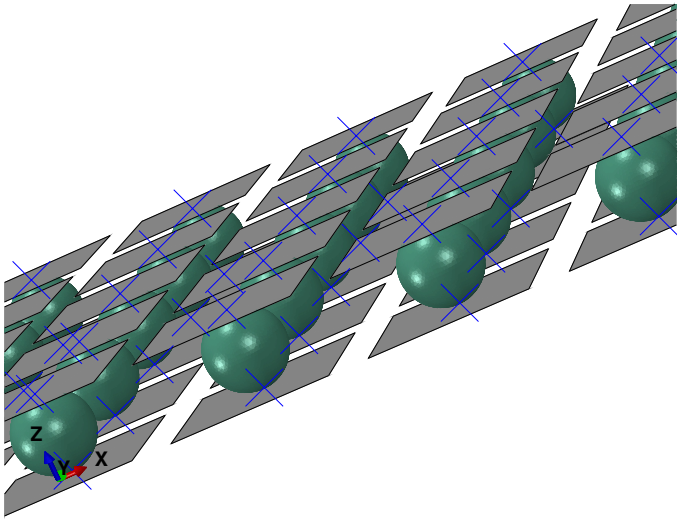


Figure 6: Pinching of multiple solid spheres.

Table 2: Performance comparison of LCP contact with traditional contact for a grid of pinched spheres.

Quantity	Single sphere		10 spheres		50 spheres		100 spheres	
	LCP approach	Traditional approach	LCP approach	Traditional approach	LCP approach	Traditional approach	LCP approach	Traditional approach
Normalized wall time (speed up factor for LCP)	1	5 (5.0)	25	54 (2.2)	2,064	358 (0.17)	14,492	553 (0.038)
Number of global assembly and solve iterations	1	7	1	7	1	7	1	7
Number of LCP iterations	134	Not applicable	1214	Not applicable	6014	Not applicable	12014	Not applicable
Model size	42,000		420,000		2,100,000		4,200,000	
Number of potential contact constraints to LCP solver	353	Not applicable	3,530	Not applicable	17,650	Not applicable	353,00	
Active constraints from the solution	187		1,870		9,350		18,700	

Keep the following additional points in mind when using the LCP approach:

- The performance gains with the LCP approach are problem dependent. In general, the number of potential contact constraints exposed to the LCP solver should be small compared to the overall problem size. In addition, the performance of the LCP solver can start to degrade when the number of potential constraints exceeds 10,000 as seen in larger contact models. The LCP approach can be beneficial if the Newton iterations using the traditional approach are expensive; for example, if you are using the direct sparse solver, costs add up quickly when dealing with a stiffness matrix from a blocky structure (see *The Sparse Solver*).

  For various mesh sizes that result in small- to medium-sized problems as in *Table 1*, direct solver costs in the traditional approach tend to dominate the LCP costs. However, for an assembly of components, the resulting stiffness matrix typically exhibits a higher degree of sparsity. The direct sparse solver costs and, thereby, the Newton iteration costs can be much smaller compared to a similarly sized blocky model. In those cases, the traditional approach is usually more efficient.
- The cost of LCP iterations also depends on how far the final solution for actual active constraints differs from the potential constraints. For example, if the active contact constraints in the final solution are a small subset of the potential constraints, more LCP iterations are necessary to arrive at the final solution than in a problem where the initial guess is close to the final solution. On the other hand, if the initial gap distance is too small, the gap distance may have to be revised and the set of potential constraints expanded. The LCP iterations in this case are repeated with the new set of potential constraints, thereby increasing the cost of the LCP approach.
- The LCP iterations may have difficulty converging in cases where the stiffness matrix in the base state shows ill-conditioning or corresponds to instabilities. It is also necessary to ensure sufficient boundary restraints to avoid stiffness matrix singularities in the base state.
- To gain a better understanding of solution accuracy and efficiency, you may choose to compare the solution to the fully nonlinear problem with the solution from the LCP approach for the types of problems you want to solve. This ensures that ignoring various nonlinearities would still provide an adequate solution. In traditional applications where nonlinear effects and friction may be important, the modeling assumptions imposed by LCP approach may be too restrictive for its effective use.

Automatic Incrementation Control

By default, Abaqus/Standard automatically adjusts the size of the time increments to solve nonlinear problems efficiently. You need to suggest only the size of the first increment in each step of the simulation, after which Abaqus/Standard automatically adjusts the size of the increments. If you do not provide a suggested initial increment size, Abaqus/Standard will attempt to apply all of the loads defined in the step in a single increment. For highly nonlinear problems Abaqus/Standard will have to reduce the increment size repeatedly to obtain a solution, resulting in wasted CPU time. It is advantageous to provide a reasonable initial increment size because only in mildly nonlinear problems can all of the loads in a step be applied in a single increment.

The number of iterations needed to find a converged solution for a time increment will vary depending on the degree of nonlinearity in the system. With the default incrementation control, the procedure works as follows. If the solution has not converged within 16 iterations or if the solution appears to diverge, Abaqus/Standard abandons the increment and starts again with the increment size set to 25% of its previous value. It then attempts to find a converged solution with this smaller time increment. If the solution still fails to converge, Abaqus/Standard reduces the increment size again. This process is continued until a solution is found. If the time increment becomes smaller than the minimum you defined or more than 5 attempts are needed, Abaqus/Standard stops the analysis.

If the increment converges in fewer than 5 iterations, this indicates that the solution is being found fairly easily. Therefore, Abaqus/Standard automatically increases the increment size by 50% if 2 consecutive increments require fewer than 5 iterations to obtain a converged solution.

While the default automatic incrementation control is suitable for most analyses, you can change all the defaults when necessary by specifying solution controls; see *Commonly Used Control Parameters* and *Time Integration Accuracy in Transient Problems*.

Automatic Stabilization of Unstable Problems

Nonlinear static problems can be unstable. Such instabilities may be of a geometrical nature, such as buckling, or of a material nature, such as material softening. If the instability manifests itself in a global load-displacement response with a negative stiffness, the problem can be treated as a buckling or collapse problem as described in *Unstable Collapse and Postbuckling Analysis*. However, if the instability is localized, there will be a local transfer of strain energy from one part of the model to neighboring parts, and global solution methods may not work. This class of problems has to be solved either dynamically or with the aid of (artificial) damping; for example, by using dashpots.

Abaqus/Standard provides an automatic mechanism for stabilizing unstable quasi-static problems through the automatic addition of volume-proportional damping to the model. The applied damping factors can be constant over the duration of a step, or they can vary with time to account for changes over the course of a step. The latter, adaptive approach is typically preferred.

Automatic Stabilization of Static Problems with a Constant Damping Factor

Automatic stabilization with a constant damping factor is triggered by including automatic stabilization in any nonlinear quasi-static procedure. Viscous forces of the form

$$F_v = c\mathbf{M}^*v$$

are added to the global equilibrium equations

$$P - I - F_v = 0,$$

where $\mathbf{M}^*$ is an artificial mass matrix calculated with unity density, c is a damping factor, $v = \Delta u/\Delta t$ is the vector of nodal velocities, and Δt is the increment of time (which may or may not have a physical meaning in the context of the problem being solved).

For the case of static stabilization the mass matrix for Timoshenko beams is always calculated assuming isotropic rotary inertia, regardless of the type of rotary inertia specified for the beam section definition (*Rotary Inertia for Timoshenko Beams*).

Automatic stabilization does not carry over automatically to subsequent steps. It needs to be declared for any step in which you want it to be active. Abaqus/Standard recalculates new values for the damping factor, based on the declared damping intensity and on the solution of the first increment of the step. Therefore, unless you specify the same damping factor directly (see *Directly Specifying the Damping Factor* below), an analysis with an unstable step may produce slightly different results from the same analysis with the original step split into two steps. Moreover, if the instabilities in the model have not subsided by the end of a step, viscous forces may be terminated abruptly or modified at the beginning of subsequent steps, potentially causing convergence difficulties if automatic stabilization is not used in the subsequent step. If such a situation arises, it is recommended that the problem be restarted with the damping factor set equal to the value chosen by Abaqus/Standard (or to the value you specified) in the previous step. This value is printed in the message (`.msg`) file for the previous step. If it is necessary to have an accurate static equilibrium solution after an instability has occurred (and the model's behavior has returned to a stable regime), the step with automatic stabilization can be followed by a step without such stabilization.

Calculating the Damping Factor Based on the Dissipated Energy Fraction

It is assumed that the problem is stable at the beginning of the step and that instabilities may develop in the course of the step. While the model is stable, viscous forces and, therefore, the viscous energy dissipated are

very small. Thus, the additional artificial damping has no effect. If a local region goes unstable, the local velocities increase and, consequently, part of the strain energy then released is dissipated by the applied damping. Abaqus/Standard can, if necessary, reduce the time increment to permit the process to occur without the unstable response causing very large displacements. Abaqus/Standard calculates and prints to the message file the damping factor, c, based on the solution of the first increment of a step. In most applications the first increment of the step is stable without the need to apply damping. The damping factor is then determined in such a way that the dissipated energy for a given increment with characteristics similar to the first increment is a small fraction of the extrapolated strain energy. The fraction is called the *dissipated energy fraction* and has a default value of 2.0×10^{-4}. If the default value for the dissipated energy fraction is used, the adaptive automatic stabilization scheme discussed in the next section will be activated automatically by default in the step.

Alternatively, you can specify the non-default dissipated energy fraction for automatic stabilization directly.

Input File Usage: Use any of the following options to specify a nondefault dissipated energy fraction:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE=*dissipated energy fraction*
**SOILS*, STABILIZE=*dissipated energy fraction*
**STATIC*, STABILIZE=*dissipated energy fraction*
**STEADY STATE TRANSPORT*, STABILIZE=*dissipated energy fraction*
**VISCO*, STABILIZE=*dissipated energy fraction*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: *any valid step type*: **Basic**: select **Specify dissipated energy fraction** from the **Automatic stabilization** field

Considerations When the First Increment Is Unstable or Singular

There are cases where the first increment is either unstable or singular (due to a rigid body mode). In such cases it is not possible to obtain a solution to the first increment without applying some damping. Therefore, some damping is already applied during the first increment. The damping factor used for the initial increment is chosen such that the average element damping matrix component, divided by the step time, is equal to the average element stiffness matrix component multiplied by the dissipated energy fraction. If the calculated strain energy change in this increment indicates that the solution without damping is stable, the damping factor is recalculated based upon the energy method described previously. However, if the strain energy change indicates that the solution is unstable or singular, the initially calculated damping factor is maintained, and a warning message is issued indicating that the amount of damping applied may not be appropriate. In many cases the amount of damping may actually be rather large, which can affect the solution in ways that are not desirable. Therefore, if the above mentioned warning message is issued, check the viscous forces (VF) and compare them with the expected nodal forces to make sure that the viscous forces do not dominate the solution. If necessary, follow the stabilized step with another step in which stabilization is not used or with a step in which a much smaller damping factor is used.

Directly Specifying the Damping Factor

You can also specify the damping factor directly. Unfortunately, it is generally quite difficult to make a reasonable estimate for the damping factor unless a value is known from the output of previous runs; the damping factor depends not only on the amount of damping but also on mesh size and material behavior.

Input File Usage: Use any of the following options to specify the damping factor directly:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE, FACTOR=*damping factor*
**SOILS*, STABILIZE, FACTOR=*damping factor*
**STATIC*, STABILIZE, FACTOR=*damping factor*

**STEADY STATE TRANSPORT*, STABILIZE, FACTOR=*damping factor*
**VISCO*, STABILIZE, FACTOR=*damping factor*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select **Specify damping factor** from the **Automatic stabilization** field

Adaptive Automatic Stabilization Scheme

As discussed above, the automatic stabilization scheme with a constant damping factor typically works well to subside instabilities and to eliminate rigid body modes without having a major effect on the solution. However, there is no guarantee that the value of the damping factor is optimal or even suitable in some cases. This is particularly true for thin shell models, in which the damping factor may be too high when a poor estimation of the extrapolated strain energy is made during the first increment. For such models you may have to increase the damping factor if the convergence behavior is problematic or to decrease the damping factor if it distorts the solution. The former case would require you to rerun the analysis with a larger damping factor, while the latter case would require you to perform postanalysis comparison of the energy dissipated by viscous damping (ALLSD) to the total strain energy (ALLIE). Therefore, obtaining an optimal value for the damping factor is a manual process requiring trial and error until a converged solution is obtained and the dissipated stabilization energy is sufficiently small.

The adaptive automatic stabilization scheme, in which the damping factor can vary spatially and with time, provides an effective alternative approach. In this case the damping factor is controlled by the convergence history and the ratio of the energy dissipated by viscous damping to the total strain energy. If the convergence behavior is problematic because of instabilities or rigid body modes, Abaqus/Standard automatically increases the damping factor. For example, the damping factor may increase if an analysis takes extra severe discontinuity or equilibrium iterations per increment or requires time increment cutbacks. On the other hand, Abaqus/Standard may reduce the damping factor automatically if instabilities and rigid body modes subside.

The ratio of the energy dissipated by viscous damping to the total strain energy is limited by an accuracy tolerance that you specify. Such an accuracy tolerance is imposed on the global level for the whole model. If the ratio of the energy dissipated by viscous damping to the total strain energy for the whole model exceeds the accuracy tolerance, the damping factor at each individual element is adjusted to ensure that the ratio of the stabilization energy to the strain energy is less than the accuracy tolerance on both the global and local element level. The stabilization energy always increases, while the strain energy may decrease. Therefore, Abaqus/Standard restricts the ratio of the incremental value of the stabilization energy to the incremental value of the strain energy for each increment to ensure that this value has not exceeded the accuracy tolerance if the ratio of the total stabilization energy to the total strain energy exceeds the accuracy tolerance. The accuracy tolerance is a targeted value and can be exceeded in some situations, such as when there is rigid body motion or when significant non-local instability occurs.

The default accuracy tolerance used by the adaptive automatic stabilization scheme is 0.05. The default tolerance is suitable for most applications, but you have the option of specifying a nondefault accuracy tolerance if necessary. If the accuracy tolerance is set equal to zero, the adaptive automatic stabilization scheme is not activated and the automatic stabilization scheme with a constant damping factor will be used in the step.

If the accuracy tolerance is not specified but the dissipated energy fraction with the default value of 2.0×10^{-4} is used, the adaptive automatic damping algorithm will be activated automatically with an accuracy tolerance of 0.05.

Input File Usage: Use any of the following options to activate adaptive automatic stabilization with the default stabilization energy tolerance:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE
**SOILS*, STABILIZE
**STATIC*, STABILIZE
**STEADY STATE TRANSPORT*, STABILIZE
**VISCO*, STABILIZE

Use any of the following options to activate adaptive automatic stabilization with a nondefault stabilization energy tolerance:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE,
ALLSDTOL=*accuracy tolerance*
**SOILS*, STABILIZE, ALLSDTOL=*accuracy tolerance*
**STATIC*, STABILIZE, ALLSDTOL=*accuracy tolerance*
**STEADY STATE TRANSPORT*, STABILIZE,
ALLSDTOL=*accuracy tolerance*
**VISCO*, STABILIZE, ALLSDTOL=*accuracy tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select an **Automatic stabilization** method: toggle on **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *accuracy tolerance*

Default Value of the Initial Damping Factor

By default, the initial value of the damping factor is typically equal to the value that would be used for automatic stabilization with a constant damping factor (see *Calculating the Damping Factor Based on the Dissipated Energy Fraction* above). In some cases additional factors that are considered with adaptive automatic stabilization cause some differences in the initial damping factor.

Specifying the Initial Damping Factor Directly

Alternatively, you can specify the initial damping factor directly. The damping factor is adjusted based on the convergence history and the accuracy tolerance through the step.

Input File Usage: Use any of the following options to specify the initial damping factor directly with the default stabilization energy tolerance:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
**SOILS*, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
**STATIC*, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
**STEADY STATE TRANSPORT*, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
**VISCO*, STABILIZE, FACTOR=*damping factor*, ALLSDTOL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: from the **Automatic stabilization** field, select **Specify damping factor:** *damping factor*: toggle on **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *maximum ratio*

Propagating the Damping Factors from the Immediately Preceding General Step into the Current Step

Adaptive automatic stabilization provides an option to propagate the damping factors from the immediately preceding general step to the subsequent steps. The default is to not propagate the damping factors from the results of the preceding general step. In this case Abaqus recalculates the initial damping factors based on the declared dissipated energy faction and on the solution of the first increment of the step, or you can specify the initial damping factors directly.

Input File Usage: Use any of the following options to indicate that the damping factors in the current step are propagated from the immediately preceding general step:

**COUPLED TEMPERATURE-DISPLACEMENT*, STABILIZE, ALLSDTOL, CONTINUE=YES
**SOILS*, STABILIZE, ALLSDTOL, CONTINUE=YES
**STATIC*, STABILIZE, ALLSDTOL, CONTINUE=YES
**STEADY STATE TRANSPORT*, STABILIZE, ALLSDTOL, CONTINUE=YES
**VISCO*, STABILIZE, ALLSDTOL, CONTINUE=YES

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select **Use damping factors from previous general step** from the **Automatic stabilization** field: **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *accuracy tolerance*

Ensuring That an Accurate Solution Is Obtained with Automatic Stabilization

Whenever automatic stabilization is applied to a problem, check the following to ensure that accurate solutions are obtained:

- For a damping factor calculated using the dissipated energy fraction, check the factor printed to the message (`.msg`) file at the end of the first increment to ensure that a reasonable amount of damping is applied. Unfortunately, the damping factor is problem dependent; therefore, you must rely on experience from previous runs.
- Compare the viscous forces (VF) with the overall forces in the analysis, and ensure that the viscous forces are relatively small compared with the overall forces in the model.
- Compare the viscous damping energy (ALLSD) with the total strain energy (ALLIE), and ensure that the ratio does not exceed the dissipated energy fraction or any reasonable amount. The viscous damping energy may be large if the structure undergoes a large amount of motion.

The automated procedure of computing damping factors works well for many applications. However, there are cases where the computed damping factor is either too small, thus not controlling the instability, or too high, thus leading to inaccurate results. These problems are more likely to occur when using a constant damping factor—the damping factor is computed in the first increment, which may not be representative of behavior in the rest of the step. For example, consider a sequentially coupled thermal-stress analysis in which a mechanical analysis reads temperatures from a previous transient thermal analysis. Typically the thermal analysis exhibits a diffusive process, where rapid changes in temperature occurs early in the analysis and minor changes in temperature occur once steady state is reached. In such a case Abaqus will compute the extrapolated strain energy based on the temperatures corresponding to the time of the first increment (in this case there may be a significant change in temperature for the first increment), thus yielding a larger than expected extrapolated strain energy. This in turn leads to a damping factor that is too large, resulting in inaccurate results.

If one of the automatic stabilization methods is not working appropriately, you can try using the other automatic stabilization method; the adaptive stabilization scheme is generally preferred. Alternatively, you can try directly specifying the damping factor.

Input File Template

The following template shows an example of specifying a gap distance to control potential constraints exposed to the LCP solver.

```
*HEADING
…
*STEP,PERTURBATION
*STATIC
Data line to define time incrementation parameters
*SOLUTION TECHNIQUE,TYPE=LCP CONTACT
Optional data line to define a gap distance and a gap scaling factor
*LOAD CASE,NAME=Case1
*CLOAD
Data line(s) to define concentrated loads for the first load case
*END LOAD CASE
**
*LOAD CASE,NAME=Case2
*BOUNDARY
Data line(s) to define boundary displacements for the second load case
*END LOAD CASE
…
*END STEP
```

Analysis Convergence Controls

In this section:

- *About Convergence and Time Integration Criteria*
- *Commonly Used Control Parameters*
- *Convergence Criteria for Nonlinear Problems*
- *Time Integration Accuracy in Transient Problems*
- *Modifying the Default Convergence Checks*

About Convergence and Time Integration Criteria

Overview

Numerous control parameters are associated with the convergence and integration accuracy algorithms in Abaqus/Standard. These parameters are assigned default values that are chosen to optimize the accuracy and efficiency of the solution for a wide spectrum of nonlinear problems.

You can change the solution control parameters, as described in the following sections:

- A brief synopsis of the more important solution control parameters, together with a description of the circumstances in which they can be used effectively, is provided in *Commonly Used Control Parameters*. This section is likely to be the most useful for the general user and should be read first.
- Abaqus/Standard incorporates an empirical algorithm designed to solve the equilibrium equations of nonlinear systems accurately and economically. The criteria used to establish convergence of nonlinear increments and the automatic adjustment of increment size based on the convergence rate are described in *Convergence Criteria for Nonlinear Problems*.
- Abaqus/Standard allows you to choose "time integration accuracy parameters" in problems that have a physical time scale. The algorithms that use these parameters for automatically controlling time increment sizes are described in *Time Integration Accuracy in Transient Problems*.

References:

Modifying the Default Solution Controls

The default values for the solution control parameters need not be adjusted for most cases. You can reset them, however, within a step definition.

Values given for the solution control parameters remain in effect for the remainder of the analysis or until they are reset.

Input File Usage: **CONTROLS*

The **CONTROLS* option can be repeated, if necessary, with different parameters.

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**

Resetting All Default Solution Controls

You can restore all solution control parameters to their default values.

Input File Usage: **CONTROLS*, RESET

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Reset all parameters to their system-defined defaults**

Commonly Used Control Parameters

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Convergence and Time Integration Criteria*
- **CONTROLS*
- *Customizing general solution controls*

Overview

Solution control parameters can be used to control:

- nonlinear equation solution accuracy and
- time increment adjustment.

These solution control parameters need not be changed for most analyses. In difficult cases, however, the solution procedure may not converge with the default controls or may use an excessive number of increments and iterations. After it has been established that such problems are not due to modeling errors, it may be useful to change certain control parameters.

This section presents a brief synopsis of the more important solution control parameters, together with a description of the circumstances in which they can be used effectively.

Values given for the solution control parameters remain in effect for the remainder of the analysis or until they are reset. You can restore all solution control parameters to their default values (see *About Convergence and Time Integration Criteria*).

Terminology

In this section the word “flux” means the variable whose discretized equilibrium is being sought and for which the equilibrium equations may be nonlinear: force, moment, heat flux, concentration volumetric flux, or pore liquid volumetric flux. The word “field” refers to the basic variables of the system, such as the components of the displacement in a continuum stress analysis or temperature in a heat transfer analysis. The superscript α refers to one such type of equation. The fields and corresponding fluxes available in Abaqus/Standard are listed in *Convergence Criteria for Nonlinear Problems*.

Defining Tolerances for Field Equations

Solution control parameters can be used to define tolerances for field equations. You can select the type of equation for which the solution control parameters are being defined, as shown in *Table 1*. The default tolerances can be reset if the analysis does not require high accuracy in the convergence criteria.

Table 1: Selecting the field equation. The input file parameters are followed by the corresponding Abaqus/CAE conditions in parentheses.

Equilibrium equation	Input file (Abaqus/CAE)	DOF
All active fields	FIELD=GLOBAL (**Apply to all applicable fields**)	all
Force and bimoment	FIELD=DISPLACEMENT (**Displacement**)	1, 2, 3, 7

Equilibrium equation	Input file (Abaqus/CAE)	DOF
Moment	FIELD=ROTATION (**Rotation**)	4, 5, 6
Heat transfer	FIELD=TEMPERATURE (**Temperature**)	11, 12, 13, ...
Hydrostatic fluid	FIELD=HYDROSTATIC FLUID PRESSURE (**Hydrostatic Fluid Pressure**)	8
Pore fluid pressure	FIELD=PORE FLUID PRESSURE (**Pore Fluid Pressure**)	8
Slurry transport	FIELD=SLURRYVF (**Slurry transport**)	31
Mass diffusion	FIELD=CONCENTRATION (**Concentration**)	11
Electrical conduction	FIELD=ELECTRICAL POTENTIAL (**Electrical Potential**)	9
Mechanism analysis (connector elements with material flow degree of freedom)	FIELD=MATERIAL FLOW (**Unsupported**)	10
Analysis containing C3D4H elements (all materials, except compressible hyperelastic elastomers and elastomeric foams).	FIELD=PRESSURE LAGRANGE MULTIPLIER (**Unsupported**)	N/A
Analysis containing C3D4H elements with compressible hyperelastic or hyperfoam materials.	FIELD=VOLUMETRIC LAGRANGE MULTIPLIER (**Unsupported**)	N/A

The most significant solution control parameters for field equation tolerances—R_n^α, C_n^α, $\tilde{q}_0^\alpha$, and $\tilde{q}_u^\alpha$—may have to be modified in cases where the residuals are large relative to the fluxes or in cases where the incremental solution is essentially zero.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Field Equations**: **Apply to all applicable fields** or **Specify individual fields**: *field*

Modifying the Residual Control

R_n^α is the convergence criterion for the ratio of the largest (scaled) residual to the corresponding average flux norm, $\tilde{q}^\alpha$, for convergence. $\tilde{q}^\alpha$ is defined in *Convergence Criteria for Nonlinear Problems*. The default value is $R_n^\alpha = 5 \times 10^{-3}$, which is rather strict by engineering standards but in all but exceptional cases will guarantee an accurate solution to complex nonlinear problems. The value for this ratio can be increased to a larger number if some accuracy can be sacrificed for computational speed.

Modifying the Solution Correction Control

C_n^α is the convergence criterion for the ratio of the largest solution correction to the largest corresponding incremental solution value. The default value is $C_n^\alpha = 10^{-2}$. In addition to sufficiently small residuals, Abaqus/Standard requires that the largest correction to the solution value be small in comparison to the largest corresponding incremental solution value. Some analyses may not require such accuracy, thus permitting this ratio to be increased. To avoid testing the magnitude of the solution correction, you can set C_n^α to 1.0.

Specifying the Average Flux

$\tilde{q}^{\alpha}$ is the value of average flux used by Abaqus/Standard for checking residuals. The default value is the time average flux calculated by Abaqus/Standard, as defined in *Convergence Criteria for Nonlinear Problems*. You may, however, define a constant value, $\tilde{q}_u^{\alpha}$, for the average flux, in which case $\tilde{q}^{\alpha} = \tilde{q}_u^{\alpha}$ throughout the step.

You may wish to use absolute tolerances for your residual checks. The absolute tolerance value is then equal to the product of the average flux, $\tilde{q}_u^{\alpha}$, and the ratio R_n^{α}.

Modifying the Initial Time Average Flux

$\tilde{q}_0^{\alpha}$ is the initial value of the time average flux for the current step. The default value is the time average flux from the previous step or 10^{-2} if this is Step 1. Redefining $\tilde{q}_0^{\alpha}$ is sometimes helpful when a coupled problem is analyzed and some of the fields in the problem are not active in the first step; for example, if a static step is carried out before a fully coupled thermal-stress step.

Redefinition of $\tilde{q}_0^{\alpha}$ can also be useful if the first step is essentially a null step; for example, in a contact problem before any contact occurs, the initial fluxes (forces) generated are zero. In such cases $\tilde{q}_0^{\alpha}$ should be given as a typical flux magnitude that will occur when field α first becomes active.

The initial value of $\tilde{q}^{\alpha}$ is retained until an iteration is completed for which $\bar{q}^{\alpha} > \epsilon^{\alpha}\tilde{q}^{\alpha}$, at which time we redefine $\tilde{q}^{\alpha} = \bar{q}^{\alpha}$. This new $\tilde{q}^{\alpha}$ can become less than $\tilde{q}_0^{\alpha}$. The criterion for zero flux compared to $\tilde{q}^{\alpha}$ is $\bar{q}^{\alpha} \le \tilde{q}^{\alpha} \cdot \epsilon^{\alpha}$ (see *Convergence Criteria for Nonlinear Problems*).

If you specify the average flux, $\tilde{q}_u^{\alpha}$, directly, the value given for $\tilde{q}_0^{\alpha}$ is ignored.

Abaqus/Standard Output

The controls in effect for an analysis are listed in the data (`.dat`) and message (`.msg`) files. Nondefault controls are marked by ***. For example, specifying the following controls:

Field Equation	R_n^{α}	C_n^{α}	$\tilde{q}_0^{\alpha}$	$\tilde{q}_u^{\alpha}$	R_P^{α}	ϵ^{α}
Displacement	0.01	1.0	10.0	–	–	1.E–4
Rotation	0.02	2.0	20.0	2.E3	–	–

would result in the following output:

```
 CONVERGENCE TOLERANCE PARAMETERS FOR FORCE
 *** CRITERION FOR RESIDUAL FORCE     FOR A NONLINEAR PROBLEM
1.000E-02
 *** CRITERION FOR DISP.    CORRECTION IN A NONLINEAR PROBLEM
1.00
 *** INITIAL VALUE OF TIME AVERAGE FORCE
10.0
     AVERAGE FORCE     IS TIME AVERAGE FORCE
     ALTERNATE CRIT. FOR RESIDUAL FORCE     FOR A NONLINEAR PROBLEM
2.000E-02
 *** CRITERION FOR ZERO FORCE     RELATIVE TO TIME AVRG. FORCE
1.000E-04
 *** CRITERION FOR RESIDUAL FORCE     WHEN THERE IS ZERO FLUX
1.000E-04
     CRITERION FOR DISP.    CORRECTION WHEN THERE IS ZERO FLUX
1.000E-03
```

```
     CRITERION FOR RESIDUAL FORCE     FOR A LINEAR INCREMENT
1.000E-08
     FIELD CONVERSION RATIO
1.00

 CONVERGENCE TOLERANCE PARAMETERS FOR MOMENT
 *** CRITERION FOR RESIDUAL MOMENT    FOR A NONLINEAR PROBLEM
2.000E-02
 *** CRITERION FOR ROTATION CORRECTION IN A NONLINEAR PROBLEM
2.00
 *** USER DEFINED VALUE OF AVERAGE MOMENT    NORM
2.000E+03
     ALTERNATE CRIT. FOR RESIDUAL MOMENT    FOR A NONLINEAR PROBLEM
2.000E-02
     CRITERION FOR ZERO MOMENT    RELATIVE TO TIME AVRG. MOMENT
1.000E-05
     CRITERION FOR RESIDUAL MOMENT    WHEN THERE IS ZERO FLUX
1.000E-05
     CRITERION FOR ROTATION CORRECTION WHEN THERE IS ZERO FLUX
1.000E-03
     CRITERION FOR RESIDUAL MOMENT    FOR A LINEAR INCREMENT
1.000E-08
     FIELD CONVERSION RATIO
1.00
```

Controlling the Time Incrementation Scheme

Solution control parameters can be used to alter both the convergence control algorithm and the time incrementation scheme. The time incrementation parameters I_0 and I_R are the most significant since they have a direct effect on convergence. They may have to be modified if convergence is (initially) nonmonotonic or if convergence is nonquadratic.

Nonmonotonic convergence may occur if various nonlinearities interact; for example, the combination of friction, nonlinear material behavior, and geometric nonlinearity may lead to nonmonotonically decreasing residuals.

Nonquadratic convergence will occur if the Jacobian is not exact, which may occur for complex material models. It may also occur if the Jacobian is nonsymmetric but the symmetric equation solver is used. In that case the unsymmetric equation solver should be specified for the step (see *Defining an Analysis*).

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**

Specifying the Equilibrium Iteration for a Residual Check

I_0 is the number of equilibrium iterations after which the check is made that the residuals are not increasing in two consecutive iterations. The default value is $I_0 = 4$. If the initial convergence is nonmonotonic, it may be necessary to increase this value.

Specifying the Equilibrium Iteration for a Logarithmic Rate of Convergence Check

I_R is the number of equilibrium iterations after which the logarithmic rate of convergence check begins. The default value is $I_R = 8$. In cases where convergence is nonquadratic and this cannot be corrected by using the

unsymmetric equation solver for the step, the logarithmic convergence check should be eliminated by setting this parameter to a high value.

Avoiding Premature Cutbacks in Difficult Analyses

Sometimes it is useful to increase both I_0 and I_R. For example, in a difficult analysis involving both friction and the concrete material model, it may be helpful to set $I_0 = 8$ and $I_R = 10$ to avoid premature cutbacks of the time increment. These two parameters can be raised to more appropriate values for severely discontinuous problems by increasing them individually.

Automatically Setting the Time Incrementation Parameters

You can automatically set the parameters described above to the values $I_0 = 8$ and $I_R = 10$. In this case any values that you specified previously for I_0 and I_R are overridden. However, if I_0 and I_R are specified multiple times in a step with different solution control settings, the last definition will be used.

Input File Usage: **CONTROLS*, ANALYSIS=DISCONTINUOUS

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**: **Discontinuous analysis**

Improving Solution Efficiency in a Problem That Involves a High Coefficient of Friction

The solution efficiency can sometimes be improved in an analysis that involves a high coefficient of friction by automatically setting the time incrementation parameters and using the unsymmetric equation solver.

Abaqus/Standard Output

The controls in effect for an analysis are listed in the data (`.dat`) and message (`.msg`) files. Nondefault controls are marked by ***. For example, specifying the time incrementation parameters I_0=7 and I_R=10 would result in the following output:

```
TIME INCREMENTATION CONTROL PARAMETERS:
*** FIRST EQUILIBRIUM ITERATION FOR CONSECUTIVE DIVERGENCE CHECK
     7
*** EQUILIBRIUM ITERATION AT WHICH LOG. CONVERGENCE RATE CHECK BEGINS
    10
     EQUILIBRIUM ITERATION AFTER WHICH ALTERNATE RESIDUAL IS USED
     9
     MAXIMUM EQUILIBRIUM ITERATIONS ALLOWED
    16
     EQUILIBRIUM ITERATION COUNT FOR CUT-BACK IN NEXT INCREMENT
    10
     MAXIMUM EQUILIB. ITERS IN TWO INCREMENTS FOR TIME INCREMENT INCREASE
     4
     MAXIMUM ITERATIONS FOR SEVERE DISCONTINUITIES
    50
     MAXIMUM ATTEMPTS ALLOWED IN AN INCREMENT
     5
     MAXIMUM DISCON. ITERS IN TWO INCREMENTS FOR TIME INCREMENT INCREASE
    50
     MAXIMUM CONTACT AUGMENTATIONS FOR *SURFACE BEHAVIOR,AUGMENTED LAGRANGE
   50
     CUT-BACK FACTOR AFTER DIVERGENCE                                0.2500
```

```
    CUT-BACK FACTOR FOR TOO SLOW CONVERGENCE                          0.5000

    CUT-BACK FACTOR AFTER TOO MANY EQUILIBRIUM ITERATIONS             0.7500

   CUT-BACK FACTOR AFTER TOO MANY SEVERE DISCONTINUITY ITERATIONS     0.2500

    CUT-BACK FACTOR AFTER PROBLEMS IN ELEMENT ASSEMBLY                0.2500
```

Activating the "Line Search" Algorithm

In strongly nonlinear problems the Newton algorithms used in Abaqus/Standard may sometimes diverge during equilibrium iteration. The line search algorithm (discussed in *Improving the Efficiency of the Solution by Using the Line Search Algorithm*) detects these situations automatically and applies a scale factor to the computed solution correction, which helps to prevent divergence. The line search algorithm is particularly useful when the quasi-Newton method (see *Solution Method*) is used.

By default, the line search algorithm is enabled only during steps where the quasi-Newton method is used. Set the maximum number of line search iterations, N^{ls}, to a reasonable value (such as 5) to activate the line search procedure or to zero to forcibly deactivate the line search.

Input File Usage: **CONTROLS*, PARAMETERS=LINE SEARCH
N^{ls}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Line Search Control**: N^{ls}

Defining Tolerances for Constraint Equations

Solution control parameters can be used to set tolerances for constraint equations. You can set strain compatibility tolerances for hybrid elements, displacement and rotation compatibility tolerances for distributing coupling constraints (specified as surface-based constraints or using DCOUP2D/DCOUP3D elements), and compatibility tolerances for softened contact. See *Convergence Criteria for Nonlinear Problems* for details.

Controlling the Solution Accuracy in Direct Cyclic Analysis

Solution control parameters can be used in direct cyclic analysis to specify when to impose the periodicity conditions and to set tolerances for stabilized state and plastic ratcheting detections.

Input File Usage: **CONTROLS*, TYPE=DIRECT CYCLIC
$I_{PI}, CR_n^{\alpha}, CU_n^{\alpha}, CR_0^{\alpha}, CU_0^{\alpha}$

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Direct Cyclic**: $I_{PI}, CR_n^{\alpha}, CU_n^{\alpha}, CR_0^{\alpha}, CU_0^{\alpha}$

Imposing the Periodicity Condition

You can specify the iteration number at which the periodicity condition is first imposed, I_{PI}. The default value is I_{PI} = 1, in which case the periodicity condition is imposed for all iterations from the beginning of an analysis. This solution control parameter rarely needs to be reset from its default value.

Defining Tolerances for Stabilized State and Plastic Ratcheting Detections

You can specify the stabilized state detection criteria, CR_n^α and CU_n^α. CR_n^α is the maximum allowable ratio of the largest (scaled) residual coefficient on any terms in the Fourier series to the corresponding average flux norm, and CU_n^α is the maximum allowable ratio of the largest correction to the displacement coefficient on any terms in the Fourier series to the largest displacement coefficient. The default values are $CR_n^\alpha = 5 \times 10^{-3}$ and $CU_n^\alpha = 5 \times 10^{-3}$. The solution converges to a stabilized state if both these criteria are satisfied.

If plastic ratcheting occurs, the shape of the stress-strain curves remains unchanged but the mean value of the plastic strain over a cycle continues to shift from one iteration to the next. In that case it is desirable to use separate tolerances for the constant term in the Fourier series to detect the plastic ratcheting.

You can also specify the plastic ratcheting detection criteria, CR_0^α and CU_0^α. CR_0^α is the maximum allowable ratio of the largest (scaled) residual coefficient on the constant term in the Fourier series to the corresponding average flux norm, and CU_0^α is the maximum allowable ratio of the largest correction to the displacement coefficient on the constant term in the Fourier series to the largest displacement coefficient. The default values are $CR_0^\alpha = 5 \times 10^{-3}$ and $CU_0^\alpha = 5 \times 10^{-3}$. Plastic ratcheting is expected if the residual coefficients and the corrections to the displacement coefficients on any of the periodic terms are within the tolerances set by CR_n^α and CU_n^α, respectively, but the maximum residual coefficient on the constant term and the maximum correction to the displacement coefficient on the constant term exceed the tolerances set by CR_0^α and CU_0^α, respectively.

Abaqus/Standard Output

The controls in effect for an analysis are listed in the data (`.dat`) and message (`.msg`) files. Nondefault controls are marked by **. For example, specifying the following controls:

I_{PI}	CR_n^α	CU_n^α	CR_0^α	CU_0^α
5	1.0E–4	1.0E–4	1.0E–4	1.E–4

would result in the following output:

```
 STABILIZED STATE AND PLASTIC RATCHETING DETECTION PARAMETERS FOR FORCE

 *** CRITERION FOR RESI. COEFF. ON ANY FOURIER TERMS
1.000E-04
 *** CRITERION FOR CORR. TO DISP. COEFF. ON ANY FOURIER TERMS
1.000E-04
 *** CRITERION FOR RESI. COEFF. ON CONSTANT FOURIER TERM
1.000E-04
 *** CRITERION FOR CORR. TO DISP. COEFF. ON CONSTANT FOURIER TERM
1.000E-04

PERIODICITY CONDITION CONTROL PARAMETER:
 *** ITERATION NUMBER AT WHICH PERIODICITY CONDITION STARTS TO IMPOSE
     5
```

Convergence Criteria for Nonlinear Problems

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Convergence and Time Integration Criteria*
- **CONTROLS*
- *Customizing general solution controls*

Overview



Warning:

The information in this section is provided for users who may wish to adjust the convergence criteria for the solution of nonlinear systems. In most cases these criteria need not be adjusted.

In nonlinear problems the governing balance equations must be solved iteratively. This section describes:

- the field equations that can be solved by Abaqus/Standard;
- the criteria used to establish convergence of each iteration during the solution;
- "severe discontinuity" iterations; and
- the line search algorithm, which can be used to improve the robustness of the Newton method.

Field Equations

Field equations can be modeled separately or fully coupled. Some fields in Abaqus/Standard can only have linear response. Each field is discretized by using basic nodal variables (the degrees of freedom at the nodes of the finite element model) such as the components of the displacement in a continuum stress analysis problem. Each field has a conjugate "flux."

Available Fields and Their Conjugate Fluxes

The fields and conjugate fluxes available in Abaqus/Standard are as follows:

Basic problem	Field	Conjugate flux
Stress analysis: force equilibrium	Displacement, $\mathbf{u}$	Force, $\mathbf{F}$
Stress analysis: moment equilibrium	Rotation, $\boldsymbol{\phi}$	Moment, $\mathbf{M}$
Stress analysis: analysis containing beams with warping	Warping, w	Bimoment, W
Heat transfer analysis	Temperature, $\boldsymbol{\theta}$	Heat flux, q
Acoustic analysis (linear only)	Acoustic pressure, u	Rate of change of fluid volumetric flux
Pore liquid flow analysis	Pore liquid pressure, u	Pore liquid volumetric flux, q
Hydrostatic fluid modeling	Fluid pressure, p	Fluid volume, V
Mass diffusion analysis	Normalized concentration, $\boldsymbol{\phi}$	Mass concentration volumetric flux, Q

Basic problem	Field	Conjugate flux
Piezoelectric analysis	Electrical potential, φ	Electrical charge, q
Electric conduction analysis	Electrical potential, φ	Electrical current, J
Mechanism analysis (connector elements with material flow degree of freedom)	Material flow	Material flux
Analysis containing C3D4H elements (all materials, except compressible hyperelastic elastomers and elastomeric foams).	Pressure Lagrange multiplier	Volumetric flux
Analysis containing C3D4H elements with compressible hyperelastic or hyperfoam materials.	Volumetric Lagrange multiplier	Pressure flux

Constraint Equations

In some cases the problem also involves constraint equations. In Abaqus/Standard the following constraints are included by using Lagrange multipliers:

Problem	Constraint variable	Constraint
Hybrid solid (except C3D4H elements)	Pressure stress	Volumetric strain compatibility
Hybrid beam	Axial force	Axial strain compatibility
Hybrid beam	Transverse shear force	Transverse shear strain compatibility
Distributing coupling	Force	Coupling displacement compatibility
Distributing coupling	Moment	Coupling rotation compatibility
Contact	Normal pressure	Surface penetration
Contact with Lagrange friction	Shear stress	Relative shear sliding

If the penalty method is used, the contact Lagrange multipliers may not be present.

Solving Coupled Field Equations

In a general problem several (possibly nonlinear) coupled field equations of types $\alpha = 1, 2, \ldots N$ must be solved and several different (possibly nonlinear) constraints of type $j = 1, 2, \ldots K$ must be satisfied simultaneously. For example, in a structural problem in which hybrid beam elements are used, $\alpha = 1$ might represent the displacement field and the equilibrium equations for the conjugate force and $\alpha = 2$ might represent the rotation field and the equilibrium equations for the conjugate moment, while $j = 1$ represents axial strain compatibility and $j = 2$ represents transverse shear strain compatibility.

Controlling the Accuracy of the Solution

The default solution control parameters defined in Abaqus/Standard are designed to provide reasonably optimal solution of complex problems involving combinations of nonlinearities as well as efficient solution of simpler nonlinear cases. However, the most important consideration in the choice of the control parameters is that any solution accepted as “converged” is a close approximation to the exact solution of the nonlinear equations. In this context “close approximation” is interpreted rather strictly by engineering standards when the default value is used, as described below.

You can reset many solution control parameters related to the tolerances used for field equations. If you define less strict convergence criteria, results may be accepted as converged when they are not sufficiently close to the

exact solution of the system. Use caution when resetting solution control parameters. Lack of convergence is often due to modeling issues, which should be resolved before changing the accuracy controls.

You can select the type of equation for which the solution control parameters are being defined; for example, you can redefine the default controls for the displacement field and warping degree of freedom equilibrium equations only. By default, the solution control parameters will apply to all active fields in the model. See *Defining Tolerances for Field Equations* for details.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*

R_n^α, C_n^α, $\tilde{q}_0^\alpha$, $\tilde{q}_u^\alpha$, R_P^α, ϵ^α, C_ϵ^α, R_l^α

C_f, ϵ_l^α, ϵ_d^α

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Field Equations**: **Apply to all applicable fields** or **Specify individual fields**: *field*

Terminology

Each field, α, that is active in the problem is tested for convergence of the field equations. The following measures are used in deciding if an increment has converged:

r_i^α

The nodal residual in the balance equation for field α and node *i*.

$\Delta u_{\max}^\alpha$

The largest change in a nodal variable of type α in the increment.

$c_{\max}^\alpha$

The largest correction to any nodal variable of type α provided by the current Newton iteration.

e^j

The largest error in a constraint of type *j*.

$\bar{q}^\alpha(t)$

The instantaneous magnitude of the flux for field α at time *t*, averaged over the entire model (spatial average flux). This average is by default defined by the fluxes that the elements apply to their nodes and any externally defined fluxes:

$$\bar{q}^\alpha(t) \stackrel{\text{def}}{=} \frac{1}{\sum_{e=1}^{E}\sum_{n_e=1}^{N_e} N_{n_e}^\alpha + N_{ef}^\alpha}\left(\sum_{e=1}^{E}\sum_{n_e=1}^{N_E}\sum_{i=1}^{N_{n_e}^\alpha}|q|_{i,n_e}^\alpha + \sum_{i=1}^{N_{ef}^\alpha}|q|_i^{\alpha,ef}\right).$$

Here, *E* is the number of elements in the model, N_e is the number of nodes in element *e*, $N_{n_e}^\alpha$ is the number of degrees of freedom of type α at node n_e of element *e*, $|q|_{i,n_e}^\alpha$ is the magnitude of the total flux component that element *e* applies at its *i*th degree of freedom of type α at its n_eth node at time *t*, N_{ef}^α is the number

of external fluxes for field α (depends on element type, loading type, and number of loads applied to an element), and $|q|_i^{\alpha,ef}$ is the magnitude of the *i*th external flux for field α.

$\tilde{q}^{\alpha}(t)$

An overall time-averaged value of the typical flux for field α so far during this step including the current increment. Normally, $\tilde{q}^{\alpha}(t)$ is defined as $\overline{q}^{\alpha}$ averaged over all the increments in the step in which $\overline{q}^{\alpha}$ is nonzero. The $\overline{q}^{\alpha}$ for the current increment is recalculated after every iteration of the current increment.

$$\tilde{q}^{\alpha}(t) \stackrel{\text{def}}{=} \frac{1}{N_t}\sum_{i=1}^{N_t}\overline{q}^{\alpha}(t|_i),$$

where N_t is the total number of increments so far in the step, including the current increment, in which $\overline{q}^{\alpha}(t|_i) > \epsilon^{\alpha}\tilde{q}^{\alpha}(t|_i)$. Here $\overline{q}^{\alpha}(t|_i)$ is the value of $\overline{q}^{\alpha}$ at increment *i* and ϵ^{α} is a small number. The default for ϵ^{α} is 10^{-5}, but in rare cases, you can change this default.

Alternatively, you can define a value for the average flux in the step, $\tilde{q}_u^{\alpha}$. In this case, $\tilde{q}^{\alpha}(t) = \tilde{q}_u^{\alpha}$ throughout the step.

At the start of the step, $\tilde{q}^{\alpha}$ is typically the value from the previous step (except for Step 1, when $\tilde{q}^{\alpha} = 10^{-2}$ by default). Alternatively, you can define an initial value for the time average flux, $\tilde{q}_0^{\alpha}$, as described in *Modifying the Initial Time Average Flux*. $\tilde{q}^{\alpha}$ retains its initial value until an iteration is completed for which $\overline{q}^{\alpha} > \epsilon^{\alpha}\tilde{q}^{\alpha}$, at which time we redefine $\tilde{q}^{\alpha} = \overline{q}^{\alpha}$. (If $\tilde{q}_u^{\alpha}$ is defined, the value defined for $\tilde{q}_0^{\alpha}$ is ignored.)

$\tilde{q}_{\max}^{\alpha}$

The time-averaged value of the largest flux corresponding to the field α during this step, excluding the current increment.

$q_{\max}^{\alpha}$

The largest flux corresponding to the field α during the current iteration.

Average Flux

The time-averaged value of the flux ($\tilde{q}^{\alpha}(t)$) is computed from the spatial average of the flux ($\overline{q}^{\alpha}(t)$) at various instants in time. In some situations where only a small part of the model is *active* (the fluxes over the rest of the model are zero or very small), the spatial average of a flux over the entire model can be very small when compared to the spatial average over the active part of the model. Over a period of time this can result in a small value for the time-averaged value of the flux and in turn may lead to a convergence criterion that is very strict by engineering standards. To avoid such an excessively strict convergence criterion, Abaqus/Standard uses an algorithm to determine the active parts of a model at any given instant.

During an iteration any flux $|q_i^{\alpha}(t)| < \epsilon_l^{\alpha}\tilde{q}_{\max}^{\alpha}$ is treated as inactive, and the corresponding degree of freedom is also marked inactive. $\tilde{q}_{\max}^{\alpha}$ is the time-averaged value of the largest flux in the model during the current step. The default value of ϵ_l^{α} is 10^{-5}; you can redefine this parameter.

At the end of an iteration the largest flux in the model during the current iteration ($q^{\alpha}_{\max}$) is compared with the time-averaged value of the largest flux ($\tilde{q}^{\alpha}_{\max}$). If $q^{\alpha}_{\max} \geq 0.1\tilde{q}^{\alpha}_{\max}$, the spatial average is computed over only the active parts of the model; if $q^{\alpha}_{\max} < 0.1\tilde{q}^{\alpha}_{\max}$, all inactive parts of the model are reclassified as active and the spatial average is computed over the entire model. The appropriate spatial average of the flux obtained in this manner is then used to compute the time-averaged flux $\tilde{q}^{\alpha}(t)$ that is used in the convergence criterion. Setting $\epsilon^{\alpha}_{i} = 0$ forces the spatial averages of a flux to be always computed over the entire model.

If you specify a value for the average flux in the step, $\tilde{q}^{\alpha}_{u}$, $\tilde{q}^{\alpha}(t) = \tilde{q}^{\alpha}_{u}$ throughout the step.

Residuals

Abaqus/Standard computes a normalized flux per node as:

$$\phi^{\alpha}_{i} = \frac{r^{\alpha}_{i}}{f^{\alpha}_{i}\tilde{q}^{\alpha}},$$

where, for field α, r^{α}_{i} is the non-normalized nodal residual, $\tilde{q}^{\alpha}$ is the time average flux, and f^{α}_{i} is a factor that depends on how many nodes of kinematic constraints depend on node *i*. $f^{\alpha}_{i} = 1$ if no nodes of kinematic constraints depend on node *i*. f^{α}_{i} increases monotonically as the number of dependent nodes for node *i* increases. For example, the number of dependent nodes for a rigid body reference node corresponds to the number of other nodes of that rigid body, which is often quite large.

Abaqus/Standard determines the maximum normalized flux, $\phi^{\alpha}_{\max}$, for field α as the maximum absolute value of ϕ^{α}_{i} among all nodes. The maximum normalized flux and the maximum flux can occur at different nodes. Abaqus/Standard typically uses

$$\phi^{\alpha}_{\max} \leq R^{\alpha}_{n}$$

as the residual check, where you can define R^{α}_{n} (by default, it is 0.005). If this inequality is satisfied, convergence is accepted if one of the following is true:

- The largest correction to the solution, $c^{\alpha}_{\max}$, is also small compared to the largest incremental change in the corresponding solution variable, $\Delta u^{\alpha}_{\max}$,
$$c^{\alpha}_{\max} \leq C^{\alpha}_{n}\Delta u^{\alpha}_{\max}.$$
- The magnitude of the largest correction to the solution that would occur with one more iteration, estimated as
$$c^{\alpha}_{est} = \frac{(\tilde{q}^{\alpha}\cdot\phi^{\alpha}_{\max})^{i}}{\min\left((\tilde{q}^{\alpha}\cdot\phi^{\alpha}_{\max})^{i-1},(\tilde{q}^{\alpha}\cdot\phi^{\alpha}_{\max})^{i-2}\right)}c^{\alpha}_{\max}$$
satisfies the same criterion:
$$c^{\alpha}_{est} \leq C^{\alpha}_{n}\Delta u^{\alpha}_{\max}.$$

You can define C^{α}_{n}; the default value is 10^{-2}.

The superscripts *i*, $i-1$, and $i-2$ refer to the iteration number, and $(r^{\alpha}_{\max})^{0}$ refers to the largest residual in field α at the start of the first iteration of the increment. See *Commonly Used Control Parameters* for more details on specifying C^{α}_{n}.

Zero Flux

In some cases there may be zero flux in the equations of type α anywhere in the model during some increments. Zero flux is defined as $\bar{q}^{\alpha} \le \epsilon^{\alpha} \tilde{q}^{\alpha}$, where, as discussed earlier, ϵ^{α} has a default value of 10^{-5} and the solution for field α is accepted if $\phi^{\alpha}_{\max} \le \epsilon^{\alpha}$. If not, $c^{\alpha}_{\max}$ is compared to $\Delta u^{\alpha}_{\max}$, and convergence for field α is accepted when $c^{\alpha}_{\max} \le C^{\alpha}_{\epsilon} \Delta u_{\max}$. The default value of C^{α}_{ϵ} is 10^{-3}; you can redefine this parameter.

Negligible Response in Some Fields

Cases may arise where more than one field is active in the model yet there is negligible response in some of the fields in some increments. If some type of physical conversion factor, f^{α}_{β}, exists between active fields α and β, $\tilde{q}^{\alpha}$ in the above paragraph can be replaced by $f^{\alpha}_{\beta} C_f \tilde{q}^{\beta}$ for those particular increments where $\tilde{q}^{\alpha}$ is deemed too small ($\bar{q}^{\alpha} \le \tilde{q}^{\alpha} < f^{\alpha}_{\beta} C_f \tilde{q}^{\beta}$) to be used realistically as part of the convergence criteria for field α. An example of f^{α}_{β} is a characteristic length to convert between force and moment.

Here, f^{α}_{β} is a factor calculated by Abaqus/Standard based on the problem definition and the fields involved and C_f is a field conversion ratio that you can define. The default value for C_f is 1.0. Currently, this concept is used only for converting between the fields associated with forces and moments, when f^{α}_{β} represents a characteristic element length.

Linear Increments

Linear cases do not require more than one equilibrium iteration per increment. If

$$\phi^{\alpha}_{\max} \le R^{\alpha}_{l}$$

for all α, the increment is considered to be linear.

You can define R^{α}_{l}; it is intended to be very small. The default value of R^{α}_{l} is 10^{-8}. Any case that passes such a stringent comparison of the largest residual with the average flux magnitude in each field is considered linear and does not require further iteration. If this requirement is satisfied at some iteration after the first, the solution is accepted without any check on the size of the correction to the solution.

Nonquadratic Convergence

In some cases quadratic convergence of the iterations is not possible because the Jacobian of the Newton scheme is approximated. If after I_P iterations the convergence rate is only linear, Abaqus/Standard uses a looser tolerance,

$$\phi^{\alpha}_{\max} \le R^{\alpha}_{p},$$

as the residual check. This tolerance modification is not applied when the quasi-Newton method is used, since it is normal for this method to require a larger number of iterations to converge.

You can define R^{α}_{p}, which is 2×10^{-2} by default. You can also define I_P (by default, $I_P = 9$; see *Controlling Iteration*).

Convergence also requires that

$$c^{\alpha}_{\max} \le C^{\alpha}_{n} \Delta u^{\alpha}_{\max}.$$

Iteration continues until both criteria are satisfied for all active fields or the increment is abandoned.

When the active field is the displacement, the convergence criterion requiring the largest displacement correction to be small relative to the maximum displacement increment ($c^{\alpha}_{\max} \le C^{\alpha}_{n} \Delta u^{\alpha}_{\max}$) is ignored when the maximum displacement increment itself is very small, as defined by $\Delta u^{\alpha}_{max} < \epsilon^{\alpha}_{d} f^{\alpha}_{\beta}$, where f^{α}_{β} is the characteristic element length. The default value for ϵ^{α}_{d} is 10^{-8}; you can redefine this parameter.

Controlling Iteration

Each increment of a nonlinear solution will usually be solved by multiple equilibrium iterations. The number of iterations may become excessive, in which case the increment size should be reduced and the increment attempted again. On the other hand, if successive increments are solved with a minimum number of iterations, the increment size may be increased. You can specify a number of time incrementation control parameters; some of them are described in this section, while the remainder are described in *Time Integration Accuracy in Transient Problems*.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
$I_0, I_R, I_P, I_C, I_L, I_G, I_S, I_A, I_J, I_T, I^c_S, I^c_J, I^c_A$
$D_f, D_C, D_B, D_A, D_S, D_H, D_D, W_G$
$D_G, D_M, D^{dyn}_M, D^{diff}_M, D_L, D_E, D_R, D_F$
D_T

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Reattempting an Increment Because of Trouble with Element or Material Calculations

Abaqus/Standard may have trouble with the element calculations because of excessive distortion in large-displacement problems or because of very large plastic strain increments. If this occurs and automatic time incrementation has been chosen, the increment will be attempted again with a time increment of D_H times the current time increment, where you can define D_H. By default, $D_H = 0.25$. If fixed time stepping has been chosen, the analysis will terminate with an error message.

Reattempting a Diverging Increment

Sometimes the increment is too large for the solution to converge at all—the initial state is outside the "radius of convergence" of the Newton method. This condition can be detected by observing the behavior of the largest residuals, $r^{\alpha}_{\max}$. In some cases these will not decrease from iteration to iteration throughout an iteration sequence that leads to convergence, but we assume that, if they fail to decrease over two consecutive iterations, the iterations should be abandoned. Thus, if

$$\min\left((\tilde{q}^{\alpha} \cdot \phi^{\alpha}_{\max})^{i}, (\tilde{q}^{\alpha} \cdot \phi^{\alpha}_{\max})^{i-1}\right) > (\tilde{q}^{\alpha} \cdot \phi^{\alpha}_{\max})^{i-2},$$

where i is the iteration counter, the iterations are abandoned. This check is first made after I_0 iterations following a solution discontinuity. You can define I_0; it must be at least 3. The default value of I_0 is 4. If fixed time stepping has been chosen, the analysis will terminate with an error message.

With automatic time stepping the increment is begun again, using a time increment of D_f times the previous attempt, where you can define D_f. By default, $D_f = 0.25$. This subdivision continues until a successful time increment is found or the minimum time increment allowed has failed, in which case the job ends with an error

message. Using the line search algorithm with $N^{ls} = 4$ sometimes helps in such cases (see *Improving the Efficiency of the Solution by Using the Line Search Algorithm*).

Reattempting an Increment When Too Many Equilibrium Iterations Are Required

In case quadratic convergence cannot be obtained, the logarithmic rate of convergence,

$$\ln\left((\tilde{q}^{\alpha} \cdot \phi^{\alpha}_{\max})^{i} / (\tilde{q}^{\alpha} \cdot \phi^{\alpha}_{\max})^{i-1}\right),$$

will often be maintained throughout the iteration process. This rate can be established during the early iterations. If convergence has not been achieved after I_R or more iterations following a solution discontinuity, if automatic time incrementation has been selected, and if the slowest convergence rate over all fields α suggests that more than I_C total iterations subsequent to the last solution discontinuity are expected to be required, the increment is begun again with a time increment of D_C times the one abandoned. If fixed time incrementation has been chosen, the iterations are continued; but if convergence is not achieved within I_C iterations after the last solution discontinuity in the increment, the analysis will terminate with an error message.

You can define the values of I_R, I_C, and D_C. By default, $I_R = 8$, $I_C = 16$, and D_C=0.5.

Increasing or Reducing the Size of the Time Increment for Efficiency

When automatic time incrementation is chosen, the effectiveness of the nonlinear equation solution is used in the selection of the next time increment (in addition to the time integration accuracy criteria discussed in *Time Integration Accuracy in Transient Problems*). If no more than I_G iterations are required in two consecutive increments, the time increment may be increased by a factor of D_D. If an increment converges but takes more than I_L iterations, the next time increment is reduced to D_B times the current time increment. You can define the values of I_G, I_L, D_D, and D_B. By default, $I_G = 4$, $I_L = 10$, $D_D = 1.5$, and $D_B = 0.75$.

Extrapolation

At each increment after the first increment of a nonlinear analysis step Abaqus/Standard estimates the solution to the increment by extrapolating the solution from the previous increment (or increments). By default, 100% linear extrapolation is used (1% for the Riks method). Extrapolation is abandoned if

$$\Delta t_i \leq D_E \Delta t_{i-1},$$

where Δt_i is the proposed new time increment, and Δt_{i-1} is the last successful time increment. You can define the value of D_E; it is 0.1 by default.

You can turn this extrapolation scheme off for a particular step—see *Defining an Analysis*.

Convergence of Strain Constraints in Hybrid Elements

Strain constraint convergence in “hybrid” elements is checked by comparing the largest error in each strain constraint, e^j, with an absolute tolerance for the corresponding error, T^j. The magnitudes of these errors are reported in the message (`.msg`) file after each iteration as “compatibility errors.” For example, the volumetric compatibility error is a measure of the accuracy with which the incompressibility constraint is satisfied. Since nonlinearity in constraint equations is generally reflected in the field equations in the same problem, no attempt is made to estimate convergence rates in these constraint equations: we assume that the measures of convergence rate in the field equations are sufficient.

You can define the T^j (T^{vol}, T^{axial}, and T^{tshear}). By default, all of the T^j = 10^{-5}.

Input File Usage: **CONTROLS*, PARAMETERS=CONSTRAINTS
T^{vol}, T^{axial}, T^{tshear}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Constraint Equations**

Severe Discontinuity Iterations

Abaqus/Standard distinguishes between regular, equilibrium iterations (in which the solution varies smoothly) and severe discontinuity iterations (SDIs) in which abrupt changes in stiffness occur. By default, Abaqus/Standard will continue to iterate until the severe discontinuities are sufficiently small (or no severe discontinuities occur) and the equilibrium (flux) tolerances are satisfied. For more information on the criteria used for the severe discontinuity checks, see *Severe Discontinuities in Abaqus/Standard*. Alternatively, Abaqus/Standard will continue to iterate until no severe discontinuities occur and the equilibrium (flux) tolerances are satisfied. This more traditional method can cause convergence difficulties if the contact conditions are only weakly determined and contact "chattering" occurs or if a large number of severe discontinuity iterations are required to settle the contact conditions.

You can define the contact and slip compatibility tolerance, the soft contact compatibility tolerance for low pressure, and the contact force error tolerance.

Input File Usage: **CONTROLS*, PARAMETERS=CONSTRAINTS
, , , T^{cont}, T^{soft}, , , T^{cfe}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Constraint Equations**

Defining the contact force error tolerance is not supported in Abaqus/CAE.

Severe Discontinuity Iterations in Implicit Dynamic Analysis

In implicit dynamic analysis, the average time of all contact changes in the increment is estimated and the time incrementation is interrupted to solve impact equations at that time. With augmented Lagrange or penalty constraint enforcement methods or with softened contact, no contact constraints are imposed when impact equations are solved. However, if the contact constraints are not satisfied within given tolerances, a severe discontinuity iteration is forced. See *Intermittent contact/impact* for details on intermittent contact in dynamic problems.

Controlling the Number of Severe Discontinuity Iterations

By default, Abaqus applies sophisticated criteria involving changes in penetration, changes in the residual force, and the number of severe discontinuities from one iteration to the next to determine whether iteration should be continued or terminated. Hence, it is in principle not necessary to limit the number of severe discontinuity iterations. This makes it possible to run contact problems that require large numbers of contact changes without having to change the control parameters. It is still possible to set a limit, I_S^c, for the maximum number of severe discontinuity iterations; by default, $I_S^c = 50$, which in practice should always be more than the actual number of iterations in an increment.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
, , , , , , , , , , I_S^c

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Controlling the Number of Severe Discontinuity Iterations When Severe Discontinuities Always Force Iterations

In this case a limit, I_S, is placed on the number of iterations caused by severe discontinuities in an increment. If more than I_S iterations are required for severe discontinuities, the increment is started over with a time increment size of D_S times the abandoned increment size (for automatic time incrementation). If fixed time incrementation was chosen, the analysis terminates with an error message. You can define the values of I_S and D_S. By default, $I_S = 12$ and $D_S = 0.25$.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
, , , , , , I_S
, , , , D_S

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Improving the Efficiency of the Solution by Using the Line Search Algorithm

Abaqus/Standard provides the option of including a "line search" algorithm. The purpose of the line search is to improve the robustness of the Newton or quasi-Newton methods. By default, the line search is active only for steps that use the quasi-Newton method. During equilibrium iterations where residuals are large, the line search algorithm scales the correction to the solution by a line search scale factor, s^{ls}. An iterative process is used to find the value of s^{ls} that minimizes the component of the residual vector in the direction of the correction vector; this component is called y^j, where j is the line search iteration number. Each line search iteration requires one pass through the Abaqus/Standard element loop but does not require any operations using the global stiffness matrix.

It is usually sufficient to determine s^{ls} only to modest accuracy. There are several controls used to limit this accuracy. A maximum of $j = N^{ls}$ line search iterations are performed. There is a limit on the allowable range of s^{ls}:

$$s^{ls}_{min} \le s^{ls} \le s^{ls}_{max}.$$

The line search ceases when

$$\left|y^j\right| \le f^{ls}_s \left|y^0\right|,$$

where y^0 is evaluated before the first equilibrium iteration. The residual reduction factor at which the line search ceases, f^{ls}_s, is typically set to a rather loose tolerance. The line search algorithm will also cease when the change in s^{ls} provided by a line search iteration is less than η^{ls} times s^{ls}.

You can define the values of N^{ls}, s^{ls}_{max}, s^{ls}_{min}, f^{ls}_{s}, and η^{ls}. By default, N^{ls} = 0 with the Newton method, and N^{ls}=5 with the quasi-Newton method. Set N^{ls} to a nonzero value to activate the line search algorithm or to zero to forcibly deactivate line search. Default values for the additional line search parameters are s^{ls}_{max} = 1.0, s^{ls}_{min} = 0.0001, f^{ls}_{s} = 0.25, and η^{ls} = 0.10. These defaults are chosen to achieve modest accuracy for the line search scale factor, while minimizing the additional cost of line search iterations. More agressive line searching can be beneficial in some simulations, especially when many nonlinear iterations and/or cutbacks are needed to resolve sharp discontinuities in the solution. In these cases you could try allowing more line search iterations (N^{ls}=10) and requiring more accuracy in the line search scale factor (η^{ls}=0.01). This may result in more line search iterations but fewer nonlinear iterations and cutbacks and an overall reduction in solution cost.

Input File Usage: **CONTROLS*, PARAMETERS=LINE SEARCH

N^{ls}, s^{ls}_{max}, s^{ls}_{min}, f^{ls}_{s}, η^{ls}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Line Search Control**

Time Integration Accuracy in Transient Problems

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Convergence and Time Integration Criteria*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Uncoupled Heat Transfer Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*
- *Rate-Dependent Plasticity: Creep and Swelling*
- **CONTROLS*
- *Customizing general solution controls*

Overview

Abaqus/Standard usually uses automatic time stepping schemes for the solution of transient problems. Factors influencing the increment size for transient problems include convergence aspects related to the degree of geometric, material, and contact nonlinearity (which also influence non-transient problems and are discussed in *Convergence Criteria for Nonlinear Problems*) and the ability of the time integration operator to accurately resolve variations in the accelerations, velocities, and displacements over an increment. This section discusses tolerance parameters and adjustments to the time increment size related to the latter aspect.

Time Incrementation Parameters and Adjustment Criteria

Table 1 lists tolerance parameters available for specific analysis procedures. Descriptions of time integrators for the transient procedure types and, in the case of implicit dynamics, discussion of additional factors influencing the time increment size related to accuracy of time integration are provided in the respective sections referenced in *Table 1*.

Table 1: Time integration accuracy measures for various procedures.

Procedure	Accuracy measure $\left(S^J\right)$	Tolerance $\left(T^J\right)$
Implicit dynamics (*Implicit Dynamic Analysis Using Direct Integration*)	Half-increment residual	Half-increment residual tolerance
Transient heat transfer analysis (*Uncoupled Heat Transfer Analysis*)	Temperature increment, $\Delta\theta$	$\Delta\theta_{max}$
Consolidation analysis (*Coupled Pore Fluid Diffusion and Stress Analysis*)	Pore pressure increment, Δu_w	Δu_w^{max}
Creep and viscoelastic material behavior (*Rate-Dependent Plasticity: Creep and Swelling*)	$\left(\dot{\bar{\varepsilon}}^{cr}\big\vert_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}\big\vert_{t}\right)\Delta t$	Creep tolerance

In any transient analysis where automatic time incrementation is used, some of these tolerances, T^J, $J = 1, 2, \ldots$, will be active. Corresponding measures of the integration accuracy, S^J, will be calculated for each increment in the step. Abaqus/Standard will use these values to adjust the time incrementation using the criteria described in this section. The smallest time increment required by all criteria is used if more than one accuracy measure is active.

Reducing the Time Increment Size

If $S^J > T^J$ for any control, *J*, that is active in the step, the time increment Δt is too large to satisfy that time integration accuracy requirement. The increment is, therefore, begun again with a time increment of

$$D_A \left(\frac{T^J}{S^J} \right) \Delta t,$$

where you can define the value of D_A. By default, D_A = 0.85.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
first data line
, , , D_A

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Increasing the Time Increment Size

If at the current time increment, Δt,

$$\Delta t \left(\frac{S^J}{\Delta t} \right)_i < W_G T^J$$

for all *J* in each of I_T consecutive increments, *i*, and if no cut-back has occurred within those increments because of nonlinearity, the next time increment will be increased to

$$\min(D_G \Delta t_p, D_M \Delta t).$$

You can define the values of I_T, W_G, and D_G. By default, I_T = 3, W_G = 0.75, and D_G = 0.8. Δt_p is the proposed new time increment, which is defined as

$$\Delta t_p = \left(\frac{T^J}{S^J / \Delta t} \right)$$

for transient heat transfer and transient mass diffusion problems and which is defined as

$$\Delta t_p = I_T \left(\frac{T^J}{\sum_{i=1}^{I_T} (S^J / \Delta t)_i} \right)$$

for other transient problems.

A limit, D_M, is placed on the time increment increase factor. The default value of D_M depends on the type of analysis:

- D_M^{dyn} = 1.25 for dynamic analysis
- D_M^{diff} = 2.0 for diffusion-dominated processes: creep, transient heat transfer, coupled temperature-displacement, soils consolidation, and transient mass diffusion
- D_M = 1.5 for all other cases

You can redefine D_M for each analysis type.

If the problem is nonlinear, the time increment may be restricted by the rate of convergence of the nonlinear equations. The time incrementation controls used with nonlinear problems are described in *Convergence Criteria for Nonlinear Problems*.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
, , , , , , , , , I_T
, , , , , , , W_G
D_G, D_M, D_M^{dyn}, D_M^{diff}

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Avoiding Small Changes to the Time Increment Size during Implicit Integration Procedures

In linear transient problems when Abaqus/Standard uses implicit integration, the system of equations must be reformed and decomposed whenever the time increment changes even though the stiffness matrix does not change. Therefore, to reduce the number of increments at which the system matrix changes, Abaqus/Standard makes use of the factor D_L, where

$$D_L = min\left(\frac{\Delta t_p}{D_M \Delta t}\right).$$

The definition of D_L results in the following inequality between the proposed and the current time increments:

$$\Delta t_p \geq D_L D_M \Delta t.$$

Based on this inequality the time increment is allowed to increase only when its value computed by the criteria described earlier in this section, or computed using the value of `PNEWDT` specified in certain user subroutines (*UMAT*, for example), is greater than or equal to $D_L D_M \Delta t$. The default value of D_L is 1.0, but you can redefine it to be a smaller number. Reducing D_L to a value less than 1.0 allows the time increment to increase by a factor that is smaller than D_M, thereby forcing a time increment change, even if the change is small. Otherwise, the solution continues with the same Δt.

Input File Usage: **CONTROLS*, PARAMETERS=TIME INCREMENTATION
first data line
second data line
, , , , D_L

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Modifying the Default Convergence Checks

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Convergence and Time Integration Criteria*
- **CONTROLS*
- *Customizing general solution controls*

Overview



Warning: The information in this section is provided for users who might want to adjust the default convergence criteria for the solution of nonlinear systems. In most cases these criteria need not be adjusted. The use of this set of features requires some experience and testing to ensure that the alternative convergence measures produce results with an acceptable level of accuracy.

You can use solution convergence checks to control the nonlinear equation solution accuracy, time incrementation, and overall solution time in Abaqus/Standard.
For most analyses, the solution convergence check parameters need not be changed . However, in certain simulations that involve strong coupling between multiple physical fields, alternative convergence checks might reduce the solution time with minimal change in solution accuracy. When these alternate convergence checks are specified, analogous alternate contact-related convergence checks are enforced.

This section summarizes the more important solution convergence parameters and describes the circumstances in which they can be used effectively.

Values given for the solution convergence parameters remain in effect for the remainder of the analysis or until they are reset. You can restore all solution control parameters to their default values (see *About Convergence and Time Integration Criteria*).

Convergence Checks

By default, Abaqus/Standard uses a combination of factors to determine whether the solution during a given increment has converged or not. The convergence checking algorithms are based upon the following:

- comparing the largest residual for a given field to the corresponding time-averaged flux, and
- comparing the largest correction for a field to the largest increment for the same field.

However, there are some subtleties within that basic framework (for example, to determine if the flux during a given increment is zero and/or whether the response within an increment is linear). For additional details, see *Analysis Convergence Controls* .

In addition, when contact is present in the model, the converged solution also has to satisfy contact penetration and slip compatibilities and contact force errors must be less than their respective tolerances.

The default convergence checks tend to work quite well in most applications, especially in the structural mechanics domain. However, in problems involving strong coupling between two or more physical fields (that is, multiphysics applications), the default criteria can be somewhat strict and conservative, possibly leading to a large number of iterations, cutbacks, and increments. For such problems, Abaqus/Standard also offers a set of alternative and relaxed convergence measures. Within a given step, these alternative convergence measures can be used globally such that a specified convergence measure is applied to all the active fields in the model or they can be applied

on an individual field basis. In the latter case, Abaqus/Standard continues to use the default convergence criteria for all the other fields. Within such a framework, Abaqus/Standard also provides the additional flexibility of changing the convergence criteria for the various active fields from one step to another.

Strict (Default) Convergence Measure

Abaqus/Standard provides default convergence measures applicable for all fields in a multiphysics simulation. However, you can optionally choose to apply the strict criteria to one or more specific fields only and to augment this strict criteria with alternative (and potentially different) criteria for each of the other fields. In addition to the default contact convergence checks, the strict convergence measure requires the following conditions to be satisfied before Abaqus/Standard accepts the incremental solution as converged:

- Ratio of the largest residual flux to the time averaged flux is less than a tolerance, and
- Ratio of the largest correction to the largest increment is less than a tolerance.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*, CONVERGENCE CHECK=STRICT

Moderate Convergence Measure

The moderate convergence measure requires the following conditions to be satisfied for the specified field for Abaqus/Standard to accept the incremental solution as converged:

- Ratio of the largest residual to the largest flux is less than a tolerance, and
- Ratio of the largest correction to the largest increment is less than a tolerance.

In addition, if contact is active in the model, Abaqus/Standard enforces the following modified contact convergence checks to accept convergence of iterations:

- Ratio of the maximum contact force error to the largest flux is less than a tolerance, and
- Ratio of the maximum contact compatibility error to the largest increment is less than a modified tolerance which is obtained by scaling the default tolerance by 10.

Compared to the strict convergence criterion, the moderate convergence criterion uses a different metric for the residual convergence. It compares the largest residual to the largest flux (instead of to the time-averaged flux, as in the strict measure) anywhere in the domain. This criterion is useful for applications when it is important to model the region with the largest flux for a particular field accurately. Relatively smaller changes in flux in other regions of the model do not affect the convergence in this approach. This measure might be a useful alternative in models for which the time-average flux is small, making it difficult for the strict measure to be satisfied.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*, CONVERGENCE CHECK=MODERATE

Relaxed Convergence Measure

The relaxed convergence measure is based only upon the largest correction to the specified field. The residual tolerances do not play any role for this criterion. Therefore, the relaxed measure accepts an incremental solution as converged when only the following criterion is satisfied:

- The ratio of the largest correction to the largest increment is less than a tolerance.

In addition, if contact is active in the model, Abaqus/Standard enforces modified contact convergence checks to accept convergence of iterations. In this case, Abaqus ignores the contact force error check and enforces only the following contact compatibility check:

- Ratio of the maximum contact compatibility error to the largest increment is less than a modified tolerance, which is obtained by scaling the default tolerance by 10.

The relaxed convergence measure ignores the residual checks. For well-posed nonlinear incremental boundary value problems with a nonsingular "stiffness matrix" (left-hand side of the linear system of equations), it is reasonable to assume that in most cases a small correction is the result of a small residual (right-hand side of the linear system of equations). For such well-posed problems, a relaxed convergence measure might prove more effective, especially when the time-averaged residual for a particular field is quite "small," and does not decrease with further iterations.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*, CONVERGENCE CHECK=RELAXED

Concept Design Convergence Measure

The concept design convergence measure treats an increment as converged if either the residual check or the correction check of the strict convergence measure is satisfied. In other words, the incremental solution is accepted if either one of the following conditions is satisfied:

- Ratio of the largest residual to the time-average flux is less than a tolerance, or
- Ratio of the largest correction to the largest increment is less than a tolerance.

In addition, if contact is active in the model, Abaqus/Standard enforces the following modified contact convergence checks to accept convergence of iterations:

- Ratio of the maximum contact force error to the time average flux is less than a tolerance, or
- Ratio of the maximum contact compatibility error to the largest increment is less than a modified tolerance, which is obtained by scaling the default tolerance by 10.

Input File Usage: **CONTROLS*, PARAMETERS=FIELD, FIELD=*field*, CONVERGENCE CHECK=CONCEPT DESIGN

Suggested Usage

In situations where the response of certain fields is mostly linear over the range of interest, you might get faster solutions using either the relaxed or the concept design measures, with minimal differences in results compared to the default (strict) approach. In situations where the flux might be relatively large in the regions of interest in a model (compared to the flux in other regions), you can use the moderate convergence approach to reach a converged solution faster.

In many cases, the relaxed convergence measure gives a sufficiently accurate solution with a reduced computation time. The concept design convergence measure is useful to arrive at quick solutions when evaluating multiple designs in the early stages of a design process. Once the initial design is refined, you can modify the convergence checks to use the default strict measure to validate the results obtained with the relaxed or the concept design approach.

Input File Template

The following multiphysics example consists of two fully coupled thermal-electrochemical-structural analysis steps. In this analysis the active fields are ion concentration, solid electric potential, fluid electric potential, displacement, and temperature. In the first step, the moderate convergence measure is applied for all five fields.

In the second step, the ion concentration and the fluid electric potential fields use the relaxed convergence measure, while the other fields use the default (strict) convergence measure.

```
*HEADING
*NODE
*ELEMENT
*MATERIAL
...

*STEP,NAME=step1
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTROCHEMICAL
*CONTROLS, PARAMETERS=FIELD, CONVERGENCE CHECK=MODERATE
...
*END STEP
***
*STEP,NAME=step2
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTROCHEMICAL
*CONTROLS, PARAMETERS=FIELD,  FIELD=ION CONCENTRATION, CONVERGENCE
CHECK=RELAXED
*CONTROLS, PARAMETERS=FIELD,  FIELD=FLUID ELECTRIC POTENTIAL, CONVERGENCE
CHECK=RELAXED
...
*END STEP
```

Analysis Techniques

In this section:

- *About Analysis Techniques*
- *Analysis Continuation Techniques*
- *Modeling Abstractions*
- *Special-Purpose Techniques*
- *Additive Manufacturing Process Simulation*
- *Adaptivity Techniques*
- *Optimization Techniques*
- *Eulerian Analysis Techniques*
- *Particle Methods*
- *Sequentially Coupled Multiphysics Analyses*
- *Co-Simulation*
- *Field Mapper*
- *Design Sensitivity Analysis*
- *Parametric Studies*

About Analysis Techniques

Overview

Abaqus provides an extensive selection of analysis techniques.

These techniques provide powerful tools for performing your analysis more efficiently and effectively.

References:

- *ALE Adaptive Meshing*
- *Continuum Particle Analyses*
- *Eulerian Analysis Techniques*
- *Importing and Transferring Results*
- *Fracture Mechanics*
- *Matrices*
- *Parametric Studies: Commands*
- *Sequentially Coupled Multiphysics Analyses*
- *Submodeling*
- *Substructures*
- *Surface-Based Fluid Modeling*
- *Symmetric Model Generation, Results Transfer, and Analysis of Cyclic Symmetry Models*
- *About User Subroutines and Utilities*

Analysis Continuation Techniques

In many cases your analysis results represent a significant investment of computational effort. As a result, you will often want to reduce computation costs by utilizing results from an analysis that has already been performed. In other cases your overall analysis history will be comprised of distinct Abaqus jobs, each representing a portion of the response history of the model. Abaqus provides the following analysis continuation techniques:

- Abaqus allows you to restart an analysis, as long as you request that certain files containing model and state data be saved in the original analysis. See *Restarting an Analysis*.
- You can perform part of an analysis with Abaqus/Standard or Abaqus/Explicit and continue the analysis with the other product. You can transfer results from Abaqus/Standard to Abaqus/Explicit, from Abaqus/Explicit to Abaqus/Standard, and from Abaqus/Standard to Abaqus/Standard. See *About Transferring Results between Abaqus Analyses*.

Modeling Abstractions

All Abaqus models involve certain abstractions. In addition to the traditional abstractions associated with the finite element method, you can include techniques in your model to obtain more cost-effective solutions. Abaqus provides the following techniques for modeling abstractions:

- You can create substructures by grouping a number of elements together and retaining only the degrees of freedom needed to interface with adjacent structures. This technique is particularly useful when a substructure is to be reused in the same analysis, in different analyses, or by different analysts. See *Using Substructures*.

- You can analyze local regions of a model in greater detail and interpolate the solution results from a larger coarser global model. See *About Submodeling*.
- You can allow for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the stiffness, mass, viscous damping, structural damping, and load vectors in a model. See *Generating Matrices as a Linear Analysis Step*.
- You can create a three-dimensional model in Abaqus/Standard by revolving various forms of axisymmetric and three-dimensional model sectors about an axis of symmetry (see *Symmetric Model Generation*). You can also transfer the solution obtained in an original axisymmetric model to the new model (see *Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*). In addition, for models that exhibit cyclic symmetry you can extract eigenmodes and perform mode-based steady-state dynamic analysis by modeling only a single repetitive sector of the model (see *Analysis of Models that Exhibit Cyclic Symmetry*).
- Using the periodic media analysis technique, you can effectively model systems that are repetitive in nature, such as manufacturing processes involving conveyor belts or continuous forming operations. See *Periodic Media Analysis*.
- You can define a complex beam cross-section, including multiple materials and complex geometry, and automatically generate beam element cross-section properties. See *Meshed Beam Cross-Sections*.
- Using the extended finite element method, you can model discontinuities, such as cracks, as an enriched feature without creating a mesh to match the geometry of the discontinuity. See *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*.

Special-Purpose Techniques

Certain analysis techniques do not fall into a general classification and are grouped here as special-purpose techniques. Abaqus provides the following special-purpose techniques:

- You can use the inertia relief technique as an inexpensive alternative to performing a full dynamic analysis on a free or partially constrained body subjected to loads derived from rigid body accelerations. See *Inertia Relief*.
- You can selectively remove, and later reintroduce, parts of a model. See *Element and Contact Pair Removal and Reactivation*.
- You can introduce small imperfections into a model, typically for postbuckling analysis. See *Introducing a Geometric Imperfection into a Model*.
- You can evaluate fracture performance through contour integral evaluation, through crack propagation modeling techniques, or by using line spring elements in conjunction with shell elements. See *About Fracture Mechanics*.
- You can model coupling between the deformation of a fluid-filled structure and the pressure exerted by a contained fluid (see *About Surface-Based Fluid Cavities*).
- In Abaqus/Explicit you can use the mass scaling technique to control the stable time increment and increase computational efficiency. See *Mass Scaling*.
- You can use selective subcycling to allow different time increments to be used for different groups of elements, which can reduce the run time for an analysis when a small region of elements in the model controls the stable time increment. See *Selective Subcycling*.
- You can use steady-state detection to detect the time in a quasi-static unidirectional Abaqus/Explicit simulation when a steady-state condition has been reached and then terminate the simulation. See *Steady-State Detection*.

Adaptivity Techniques

Adaptivity techniques enable modification of your mesh to obtain a better solution. Abaqus provides the following adaptivity techniques:

- You can use ALE adaptive meshing to control mesh distortion or to model material loss. See *About ALE Adaptive Meshing*.
- You can use adaptive remeshing with Abaqus/Standard and Abaqus/CAE to iteratively improve your mesh to obtain a more accurate solution. See *About Adaptive Remeshing*.
- You can use mesh-to-mesh solution mapping as part of a mesh replacement strategy for distortion control. See *Mesh-to-Mesh Solution Mapping*.
- You can perform fatigue crack growth with adaptive remeshing in Abaqus/Standard. See *Fatigue Crack Propagation with Adaptive Remeshing*.

See *About Adaptivity Techniques* for a comparison of the adaptivity methods.

Optimization Techniques

You can use structural optimization, an iterative process that helps you refine your designs, to perform topology and shape optimization. In Abaqus/CAE you create the model to be optimized and define, configure, and execute the structural optimization. See *About Structural Optimization*.

Eulerian Analysis

You can use Abaqus/Explicit to simulate extreme deformation, up to and including fluid flow, in an Eulerian analysis. Eulerian materials can be coupled to Lagrangian structures to analyze fluid-structure interactions. See *Eulerian Analysis*.

Particle Methods

Using the smoothed particle hydrodynamics technique, you can model violent free-surface fluid flow (such as wave impact) and extremely high deformation/obliteration of solid structures (such as ballistics). See *Smoothed Particle Hydrodynamics*.

Using the discrete element technique, you can model particulate media and perform analyses such as granular material mixing or segregation, transport, and deposition of particulate materials. See *Discrete Element Method*.

Sequentially Coupled Multiphysics Analyses

In Abaqus/Standard you can perform sequentially coupled multiphysics analyses when the coupling between one or more of the physical fields in a model is only important in one direction. See *Sequentially Coupled Multiphysics Analyses*.

Co-Simulation

You can use the co-simulation technique for run-time coupling of two Abaqus analyses or of Abaqus with third-party analysis programs to perform multiphysics simulation. See *About Co-Simulation*.

Extending Abaqus Analysis Functionality

You can use the flexibility of user subroutines to increase the functionality of Abaqus. See *About User Subroutines and Utilities*.

Design Sensitivity Analysis

You can use design sensitivity analysis (DSA) techniques to determine sensitivities of responses with respect to specified design parameters. You can use these techniques for design studies within Abaqus/Standard or in conjunction with third-party design optimization tools. See *Direct Design Sensitivity Analysis*.

Parametric Studies

You can use parametric studies to perform multiple analyses in which you can systematically vary modeling parameters that you define. See *Scripting Parametric Studies* and *Parametric Studies: Commands*.

Availability of Analysis Techniques

The availability of the analysis techniques provided in Abaqus is summarized in *Table 1*. In addition, adaptive remeshing and optimization techniques are available in Abaqus/CAE (see *Adaptive Remeshing* and *About Structural Optimization*).

Table 1: Availability of analysis techniques in Abaqus.

Technique	Abaqus/Standard	Abaqus/Explicit
Restarting an Analysis	√	√
Importing and Transferring Results	√	√
Substructures	√	
Submodeling	√	√
Matrices	√	
Symmetric Model Generation, Results Transfer, and Analysis of Cyclic Symmetry Models	√	
Periodic Media Analysis		√
Meshed Beam Cross-Sections	√	√
Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method	√	
Applying Cohesive Material Concepts to XFEM-Based Cohesive Behavior	√	
Applying the VCCT Technique to the XFEM-Based LEFM Approach	√	
Contact Interaction of Cracked Element Surfaces	√	
Defining the Constitutive Response of Fluid Flow within the Cracked Element Surfaces	√	
Modeling Slurry Flow	√	
Reduced Basis Generation	√	
Inertia Relief	√	

Technique	Abaqus/Standard	Abaqus/Explicit
Element and Contact Pair Removal and Reactivation	√	
Progressive Element Activation	√	
Introducing a Geometric Imperfection into a Model	√	√
Fracture Mechanics	√	
Surface-Based Fluid Modeling	√	√
Mass Scaling		√
Selective Subcycling		√
Steady-State Detection		√
CZone Analysis		√
ALE Adaptive Meshing	√	√
Mesh-to-Mesh Solution Mapping	√	
Fatigue Crack Propagation with Adaptive Remeshing	√	
Eulerian Analysis Techniques		√
Discrete Element Method		√
Continuum Particle Analyses		√
Particle Generator		√
Lumped Kinetic Molecular Method		√
Sequentially Coupled Multiphysics Analyses	√	
About Co-Simulation	√	√
Field Mapper	√	√
Direct Design Sensitivity Analysis	√	
Adjoint Design Sensitivity Analysis Overview	√	
Scripting Parametric Studies	√	√
Fatigue Crack Propagation with Adaptive Remeshing	√	

Analysis Continuation Techniques

In this section:

- *Restarting an Analysis*
- *Importing and Transferring Results*

Restarting an Analysis

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Output*
- *Loads*
- **RESTART*
- *Restarting an analysis*

Overview

When you run an analysis, you can write the model definition and state to the files required for restart.

Scenarios for using the restart capability include:

Continuing an interrupted run

If an analysis is interrupted by a computer malfunction, the Abaqus restart analysis capability allows the analysis to complete as originally defined.

Continuing with additional steps

After viewing results from a successful analysis, you might decide to append steps to the load history.

Changing an analysis

Sometimes, having viewed the results of the previous analysis, you might want to restart the analysis from an intermediate point and change the remaining load history data in some manner. In addition, you might want to add additional steps to the load history if the previous analysis completed successfully.

About Output describes the process of obtaining results output from an Abaqus/Standard restart file.

Writing Restart Files

If you want to be able to restart an analysis, you must request restart output. This output will be written to files that can be used to restart the analysis. If you do not request that restart data be written, restart files will not be created in Abaqus/Standard, while in Abaqus/Explicit a state file will be created with results at only the beginning and end of each step.

In Abaqus/Standard these files are the restart (*job-name*`.res`; file size limited to 16 gigabytes), analysis database (`.mdl` and `.stt`), part (`.prt`), output database (`.odb`), and linear dynamics and substructure database (`.sim`) files. In Abaqus/Explicit these files are the restart (*job-name*`.res`; file size limited to 16 gigabytes), analysis database (`.abq`, `.mdl`, `.pac`, and `.stt`), part (`.prt`), selected results (`.sel`), and output database (`.odb`) files. These files, collectively referred to as the restart files, allow an analysis to be completed up to a certain point in a particular run and restarted and continued in a subsequent run. The output database file only needs to contain the model data; results data are not required and can be suppressed.

You can control the amount of data written to the restart files, as described below. The amount of data written to the restart file can be changed from step to step if you include the restart request in each step definition.

Restart information is not written during the following linear perturbation steps:

- *Static Stress Analysis* (perturbation)
- *Eigenvalue Buckling Prediction*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Complex Eigenvalue Extraction*
- *Transient Modal Dynamic Analysis*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*
- *Response Spectrum Analysis*
- *Random Response Analysis*
- *Eddy Current Analysis*

Input File Usage: Use the following option to request that restart data be written for an analysis:

**RESTART*, WRITE

The **RESTART*, WRITE option can be used as either model data or history data.

Abaqus/CAE Usage: Step module: **Output->Restart Requests**

In Abaqus/CAE restart requests are always associated with a particular step; you cannot define a restart request for the entire analysis. Restart requests are created by default for every step; restart requests for Abaqus/Standard have a default frequency of 0, while restart requests for Abaqus/Explicit steps have a default number of intervals of 1.

Controlling the Frequency of Output to the Restart Files

You can specify the frequency at which data will be written to the Abaqus/Standard restart file and the Abaqus/Explicit state files. The variables to be written cannot be specified; a complete set of data is written each time. Therefore, the restart files can be quite large unless you control the frequency with which restart information is written. If restart information is requested for an Abaqus/Standard analysis at exact time intervals, Abaqus/Standard will obtain a solution each time data are written. In this case if the frequency of output to the restart file is high, the number of increments and, consequently, the computational cost of the analysis might increase considerably.

Specifying the Frequency of Output to the Abaqus/Standard Restart File in Increments

By default, Abaqus/Standard will write data to the restart file after each increment at which the increment number is exactly divisible by a user-specified frequency value, N, and at the end of each step of the analysis (regardless of the increment number at that time). In a direct cyclic or a low-cycle fatigue analysis, Abaqus/Standard will write data to the restart file only at the end of a loading cycle; therefore, Abaqus/Standard will write data to the restart file after each iteration (or cycle in a low-cycle fatigue analysis) at which the iteration number (or cycle number in a low-cycle fatigue analysis) is exactly divisible by N and at the end of each step of the analysis.

Input File Usage: **RESTART*, WRITE, FREQUENCY=*N*

By default, N=1.

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *N* in the **Frequency** column for each step

By default, *N*=0 (no restart information is written).

Specifying the Frequency of Output to the Abaqus/Standard Restart File in Time Intervals

Abaqus/Standard can divide the step into a user-specified number of time intervals, *n*, and write the results at the end of each interval, for a total of *n* points for the step. If *n* is specified, by default data will be written to the results file at the exact times calculated by dividing the step into *n* equal intervals. Alternatively, you can choose to write the information at the increment ending immediately after the time dictated by each interval.

You can specify the frequency of restart output in time intervals only for the procedures listed in *Table 1*. In addition, this capability is not supported for linear perturbation analyses.

Table 1: List of Abaqus/Standard procedures that support restart at time intervals.

Procedure	Time incrementation	Restart at exact time intervals	Restart at approximate time intervals
Static Stress Analysis (except if the Riks method is used)	Automatic	√	√
	Fixed	—	√
Implicit Dynamic Analysis Using Direct Integration	Automatic	√	√
	Fixed	—	√
Uncoupled Heat Transfer Analysis (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
Mass Diffusion Analysis (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
Coupled Pore Fluid Diffusion and Stress Analysis (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
Fully Coupled Thermal-Stress Analysis	Automatic	√	√
	Fixed	—	√
Fully Coupled Thermal-Electrical-Structural Analysis	Automatic	√	√
	Fixed	—	√
Coupled Thermal-Electrical Analysis (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√

Procedure	Time incrementation	Restart at exact time intervals	Restart at approximate time intervals
Steady-State Transport Analysis	Automatic	√	√
	Fixed	—	√
Subspace-Based Steady-State Dynamic Analysis	Fixed	—	√
Quasi-Static Analysis	Automatic	√	√
	Fixed	—	√

Input File Usage: Use the following option to request results at the exact time intervals:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=YES

Use the following option to request results at the increments ending immediately after each time interval:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=NO

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *n* in the **Intervals** column; toggle on the **Time Marks** column for each step if you want the results written at the exact time intervals

Time Incrementation

If the output frequency is specified in terms of the number of intervals, Abaqus/Standard will adjust the time increments to ensure that data are written at the exact time points specified. In some cases Abaqus might use a time increment smaller than the minimum time increment allowed in the step in the increment directly before a time point. However, Abaqus will not violate the minimum time increment allowed for consolidation, transient mass diffusion, transient heat transfer, transient couple thermal-electrical, transient coupled temperature-displacement, and transient coupled thermal-electrical-structural analyses. For these procedures if a time increment smaller than the minimum time increment is required, Abaqus will use the minimum time increment allowed in the step and will write restart data at the first increment after the time point.

When the output frequency is specified in terms of the number of intervals, the number of increments necessary to complete the analysis might increase, which might adversely affect performance.

Specifying the Frequency of Output to the Abaqus/Explicit State File

Abaqus/Explicit will divide the step into a user-specified number of time intervals, *n*, and write the results at the beginning of the step and at the end of each interval, for a total of *n*+1 points for the step, with the last point matching the end of the step. By default, the results will be written to the state file at the increment ending immediately after the time dictated by each interval. Alternatively, you can choose to write the results at the exact times calculated by dividing the step into *n* equal intervals.

If a problem precludes the analysis from continuing to completion, such as if an element becomes excessively distorted, Abaqus/Explicit will attempt to save the last completed increment in the state file.

Alternatively, you can specify the number of intervals as zero to turn off all restart frame output. This setting can be used to reduce the disk usage if there is no continuation after the current analysis.

Input File Usage: Use the following option to request results at the increments ending immediately after each time interval:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=NO

Use the following option to request results at the exact time intervals:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=YES

By default, *n*=1.

Use the following option to turn off all restart frame output:

**RESTART*, WRITE, NUMBER INTERVAL=0

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *n* in the **Intervals** column; toggle on the **Time Marks** column for each step if you want the results written at the exact time intervals

By default, *n*=1.

Turning off all restart frame output is not supported in Abaqus/CAE.

Synchronizing Restart Information Written in a Co-Simulation

Restart output must be synchronized between co-simulation analyses for a co-simulation restart to be successful. You should request that restart data are written at the co-simulation target times when both solutions are considered at equilibrium. The SIMULIA Co-Simulation Services supports a capability for the solvers participating in a co-simulation to synchronize writing restart data at target times. Any solver participating in the co-simulation can request that a restart frame is saved at an upcoming target time. This restart synchronization capability is supported only when the co-simulation participants support the option to synchronize.

You can set the number of time intervals, *n*, equal to 1 to indicate the co-simulation participants dictate when Abaqus writes restart data. In this case, restart data are always written at the end of the analysis step. You can set *n* to a value larger than 1 to divide the analysis step into *n* intervals to write restart data. The frequency of writing restart data is honored; however, the exact times depend on upcoming co-simulation target times.

You cannot request restart at specific time points in a co-simulation step.

Input File Usage: Use the following option to activate the restart synchronization capability and let the co-simulation participants determine when to synchronize restart data:

**RESTART*, WRITE, NUMBER INTERVAL=1

Use the following option to activate the restart synchronization capability and let Abaqus determine when to synchronize restart data:

**RESTART*, WRITE, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *n* in the **Intervals** column

Controlling the Precision of Output to the Abaqus/Explicit State File

By default, Abaqus/Explicit writes to the state file in double precision when the analysis is run in double precision. Alternatively, you can choose to write data to the state file in single precision if you want to reduce the size of the state file. This option might cause noisy results between step boundaries or for the first step of a restart analysis. If Abaqus/Explicit is run in single precision, this control parameter is ignored and single precision is always used.

Input File Usage: **RESTART*, WRITE, SINGLE

Abaqus/CAE Usage: Single precision state file output is not supported by Abaqus/CAE.

Overlaying Results in the Restart Files

For an Abaqus/Standard or Abaqus/Explicit analysis, you can specify that only one increment (or one iteration in the case of a direct cyclic analysis) per step should be retained in the Abaqus/Standard restart file or Abaqus/Explicit state file, thus minimizing the size of the files. As the data are written, they overlay the data from the previous increment (or iteration), if any, written for the same step. You can specify whether or not the data should be overlaid for each step individually. Since in Abaqus/Explicit the results are written by default only at the end of the step, it is recommended to overlay the data in conjunction with specifying a number of time intervals at which data are written; in this way, the data in the restart file are advanced as dictated by the number of intervals used.

To protect you from losing data if your system crashes, when Abaqus/Standard writes a frame from a given increment, it does not strictly overwrite the frame from the last saved increment. Instead, it always keeps a reserve frame and only frees a given saved frame for overwriting when the next frame is secured on the file. This reserve frame is not deleted unless the space is required for later increments. This process produces a bonus frame in the last step of an analysis if overlaying is occurring in that step and if the analysis completes successfully; users will observe that the penultimate restart frame is also retained for the last step, even though overlay is being used.

The advantage of overlaying the restart data is that it minimizes the space required to store the restart files.

Input File Usage: Use the following option in Abaqus/Standard:

**RESTART*, WRITE, OVERLAY

Use the following option in Abaqus/Explicit:

**RESTART*, WRITE, OVERLAY, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: click to check the **Overlay** column for each step

Restarting an Analysis

You restart (continue) an analysis by specifying that the restart or state, analysis database, and part files created by the original analysis be read into the new analysis. The restart files must be saved on completion of the first job. In Abaqus/Explicit the package (`.pac`) file and the selected results (`.sel`) file are also used for restarting an analysis and must be saved on completion of the first job. Since restart files can be very large, sufficient disk space must be provided (in Abaqus/Standard the analysis input file processor estimates the space that is required for the restart file).

You can specify the point at which the analysis is continued in the new run, as discussed below.

An Abaqus/Standard analysis cannot be restarted from the linear perturbation steps listed in *Writing Restart Files*. An Abaqus/Explicit analysis cannot be restarted from an anneal step.

In addition, if an Abaqus/Standard or Abaqus/Explicit analysis is terminated abruptly by an operating system command or due to a power failure, it is unlikely that the job can be recovered or restarted. In this situation, files that are open during the analysis process are not closed properly, which might result in loss of data and incomplete files.

Input File Usage: Use the following option to restart an analysis:

**RESTART*, READ

When the READ parameter is included, the **RESTART* option must appear as model data. It is normally the first option in the input file after the **HEADING* option.

Abaqus/CAE Usage: Job module: job editor: toggle on **Restart** as the **Job Type**

Identifying the Analysis to Be Restarted

In an Abaqus/Standard restart analysis you must specify the name of the restart file that contains the specified step and increment, iteration (for a direct cyclic analysis), or cycle (for a low-cycle fatigue analysis). In an Abaqus/Explicit restart analysis you must specify the name of the state file that contains the specified step and interval.

Abaqus issues an error message if the step and increment, iteration, cycle, or interval number at which restart is requested do not exist in the specified restart or state file.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **oldjob=***oldjob-name*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job** and enter the *oldjob-name*

Specifying the Restart Point

You can specify the point (step and increment, iteration, cycle, or interval) in the previous analysis from which to restart. Truncating a step in the previous analysis when you restart is discussed below.

Specifying the Restart Point for an Abaqus/Standard Analysis (Except When Restarting from a Direct Cyclic or a Low-Cycle Fatigue Analysis)

An Abaqus/Standard analysis restarted from any analysis other than a direct cyclic or a low-cycle fatigue analysis will continue the analysis immediately after the user-specified step and increment. If you do not specify a step or increment, the analysis will restart at the last available step and increment found in the restart file.

Input File Usage: **RESTART*, READ, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *increment*

Specifying the Restart Point for an Abaqus/Standard Analysis Restarted from a Direct Cyclic Analysis

An Abaqus/Standard analysis restarted from a previous direct cyclic analysis can be restarted only from the end of a loading cycle. In this case you should specify the step and iteration number at which the new analysis will be resumed.

In a direct cyclic analysis that has not reached a stabilized cycle on restart, you can increase the number of iterations or Fourier terms, thus allowing continuation of an analysis (see *Direct Cyclic Analysis*).

Input File Usage: **RESTART*, READ, STEP=*step*, ITERATION=*iteration*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *iteration*

Specifying the Restart Point for an Abaqus/Standard Analysis Restarted from a Low-Cycle Fatigue Analysis

An Abaqus/Standard analysis restarted from a previous low-cycle fatigue analysis can be restarted only from the end of a loading cycle. In this case you should specify the step and cycle number at which the new analysis will be resumed.

Input File Usage: **RESTART*, READ, STEP=*step*, CYCLE=*cycle*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *cycle*

Specifying the Restart Point for an Abaqus/Explicit Analysis

An Abaqus/Explicit restart analysis will continue the analysis immediately after the user-specified step and interval. You must specify the step from which an Abaqus/Explicit restart analysis will continue. If you do not specify an interval from which to restart or that the current step should be terminated at a specified interval, the analysis is restarted from the last interval available in the state file for the specified step.

Input File Usage: **RESTART*, READ, STEP=*step*, INTERVAL=*interval*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *interval*

Continuing an Analysis without Changes

To continue an analysis without changes, only the steps subsequent to the step at which restart is being made should be defined in the restart analysis. All other information has been saved to the restart files. This feature cannot be used for an Abaqus analysis that uses the co-simulation technique.

Continuing an Abaqus/Standard Analysis without Changes

In Abaqus/Standard, in cases where restart is being performed simply to continue a long step (which might have been terminated because the time limit for the job was exceeded, for example), the data for the restart run might simply consist of the request to read restart data from another analysis.

Input File Usage: **RESTART*, READ

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**

Continuing an Abaqus/Explicit Analysis without Changes

In Abaqus/Explicit, in cases where restart is being performed simply to continue a long step (which might have been terminated because a CPU time limit was exceeded, for example), do not use a restart analysis; instead, use a recover analysis. In this case, no data are needed (unless user subroutines are being used).

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **recover**

Abaqus/CAE Usage: Job module: job editor: toggle on **Recover (Explicit)** as the **Job Type**

Truncating a Step

You can truncate an analysis step prior to its completion when you restart the analysis. For example, by default, if the previous analysis is an Abaqus/Standard procedure and you specify that the restart point is Step *p*, the restart analysis will restart from the last saved increment of Step *p* and continue the step to completion. However, if you specify that the restart point is increment *n* of Step *p* and that the step should be terminated before restart, the restart analysis will restart from increment *n* of Step *p*, end Step *p* at that point, and continue with newly defined steps. In this case the step from which the analysis is being restarted will be truncated at the time of restart, regardless of the step end time that had been given in the previous analysis. Thus, the step is considered to be completed even though all of the loading might not have been applied. Continuation of the analysis will be defined by history data provided in the restart run.

When you truncate an analysis step in an Abaqus/Explicit restart analysis, you must specify the interval after which the analysis should be restarted.

If the step from which the restart is being made completed normally, you can truncate the step to restart within the step so that you can request additional output, write to the restart file with a higher frequency, etc. In Abaqus/Explicit it might be necessary to truncate an analysis step when an unforeseen event occurs within a step; for example, if contact surface definitions require modification due to unforeseen displacements. If the step from which the restart is being made completed normally and the restart is being made from the last increment, iteration, or interval, truncating the analysis step will have no effect.

If the restart is being made from a job that was truncated by the operating system (for example, because of insufficient disk space, run-time limit exceeded, etc.), you will usually not choose to truncate the analysis step, so that the old step will first be completed before a new step—if any exists—is started. If restart is being made from the end of a step that terminated prematurely inside Abaqus (for example, because it ran out of increments or it failed to converge), you must truncate the step and include a new step definition. If you do not truncate the step, Abaqus will try to continue the old step on restart and will terminate the analysis in the same manner as before.

Input File Usage: Use the following option in Abaqus/Standard to restart from any analysis step other than a direct cyclic step:

**RESTART*, READ, STEP=*p*, INC=*n*, END STEP

Use the following option in Abaqus/Standard to restart from a direct cyclic analysis step:

**RESTART*, READ, STEP=*p*, ITERATION=*n*, END STEP

Use the following option in Abaqus/Standard to restart from a low-cycle fatigue analysis step:

**RESTART*, READ, STEP=*p*, CYCLE=*n*, END STEP

Use the following option in Abaqus/Explicit:

**RESTART*, READ, STEP=*p*, INTERVAL=*n*, END STEP

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*model_name***: **Restart**: toggle on **Read data from job**; **Step name:** *step*; toggle on **Restart from increment, interval, iteration, or cycle**, enter the *increment*, *interval*, *iteration*, or *cycle*; and toggle on **and terminate the step at this point**

Amplitude References

Care should be taken if loads and boundary conditions refer to amplitude curves (*Amplitude Curves*). If the amplitude is given in terms of total time, the loads and boundary conditions will continue to be applied according to the amplitude definition. However, if the amplitude is given in terms of step time (default), the loads and boundary conditions will be held constant at their values at the time the step is terminated.

Temperatures, field variables, and mass flow rates applied in the old step will remain in the new step if they are not redefined. If an amplitude curve was not specified, these quantities will continue to be applied according to the default amplitude for the procedure.

Automatic Stabilization in Abaqus/Standard

In Abaqus/Standard care should be exercised when automatic stabilization is active at the point at which a step is truncated. This might happen either in the middle of quasi-static procedures using automatic stabilization (see *Solving Nonlinear Problems*) or during contact analyses using automatic viscous damping (see *Generally Applicable Contact Controls in Abaqus/Standard*). In such cases viscous forces might be present, which will not be carried over to the subsequent step, therefore causing convergence difficulties.

In the case of quasi-static procedures using automatic stabilization, it is recommended that the stabilization continue to be enforced during the following step and that you specify the damping factor directly, using the last value printed out by Abaqus/Standard in the message file. In the case of automatic viscous damping in a contact pair when contact has not yet been fully established, it is recommended that the damping be applied again, although there is no guarantee that the amount of damping applied will be the same as in the original step.

Choosing the Initial Time Increment for an Abaqus/Standard Restart Analysis

In Abaqus/Standard take care in choosing the time period and initial time increment for the new step if the previous step was truncated. In transient analyses the initial time increment for the new step should be similar to the time increment that was used at the point of restart in the old step. In quasi-static analyses choose the

initial time increment of the new step so that the increments in loads or prescribed boundary conditions are similar to those at the point of restart in the old step.

In a nonlinear analysis the increment of load applied in the first increment of the restart run should be similar to that applied in the last converged increment of the previous run. Let

Δl_1

= the load to be applied in the first increment of the restart run,

l_{rem}

= the remaining load to be applied in the restart run,

Δt_1

= the initial time increment for the restart run, and

t_{step}

= the total step time for the first step of the restart run.

The following equation can then be used to determine the initial time increment for the restart run:

$$\Delta t_1 = \frac{\Delta l_1}{l_{\text{rem}}} t_{\text{step}}.$$

Example

Suppose an Abaqus/Standard job stopped running because it reached the maximum number of increments specified for the step. The original input file was as follows:

```
*HEADING
 …
*STEP, INC=4
*STATIC, DIRECT
 0.1, 1.0
*CLOAD
 1, 2, 20.0
*RESTART, WRITE, FREQUENCY=2
*END STEP
```

This run ended at Step 1, increment 4 with a load of 8.0 applied. The following input file could be used to restart this job and to complete the loading:

```
*HEADING
*RESTART, READ, STEP=1, INC=4, END STEP
*STEP, INC=120
*STATIC, DIRECT
 0.1, 0.6
*CLOAD
 1, 2, 20.0
*END STEP
```

Notice that the concentrated load applied is the same as in the previous step.

In this example, assume that a load increment of 2.0 was applied in the last converged increment of the previous run. Therefore, the initial time increment for the restart run is chosen such that the load increment applied during the first increment is also 2.0. The remaining load to be applied in the restart run is 12.0 (20.0 total − 8.0 applied

in the previous run). Substitution into the equation for the initial time increment yields $\Delta t_1 = t_{step}/6$. The step time for the first step of the restarted job, t_{step}, is chosen to be 0.6 so that the total accumulated time is 1.0 when the applied load is 20.0 (at the end of the step). Thus, the initial time increment for the restart run, Δt_1, is set equal to 0.1.

Supplying Additional Data in the Restart Analysis

It is possible to define steps subsequent to the step at which restart is being made. It is also possible to supply new amplitude definitions, new surfaces, new node sets, and new element sets during the restart analysis. Existing sets cannot be modified.

In Abaqus/Standard additional surfaces defined in the model part of a restart analysis have the restriction that they can be referenced only from surface-based loading definitions (see *Loads*) or output requests for user-defined surface sections (see *Output to the Data and Results Files*).

Example

For example, suppose a one-step Abaqus/Explicit job stopped prior to completion because a CPU time limit was exceeded and you have decided that a second step should be added with new boundary condition definitions. The following input file could be used to restart this job, complete the remaining part of Step 1, and complete Step 2:

```
*HEADING
*RESTART, READ, STEP=1
**
** This defines Step 2
**
*STEP
*DYNAMIC, EXPLICIT
 , .003
*BOUNDARY, OP=NEW
 …
*END STEP
```

Continuation of Optional History Data in Restart Analyses

The rules governing the continuation of optional analysis history data—loading, boundary conditions, output controls, and auxiliary controls (see *Abaqus Model Definition*)—are the same for the steps defined in the restart analysis and the original analysis. For a discussion of the rules governing the continuation of optional history data, see *Defining an Analysis*.

Prescribing Predefined Fields in the Restart Analysis

It is possible to prescribe predefined fields (see *Predefined Fields*) in the restart analysis.

To specify predefined temperatures or field variables in an Abaqus/Standard restart analysis, the corresponding predefined field must have been specified in the original analysis as initial temperatures or field variables (*Initial Conditions*) or as predefined temperatures or field variables (*Predefined Fields*).

Restarting with User Subroutines

User subroutines are not written to the Abaqus/Standard restart file or to the Abaqus/Explicit state file. Therefore, if the original analysis contained any user subroutines, these subroutines must be included again in the restart run or when recovering additional results output from restart data (see *Recovering Additional Results Output from Restart Data in Abaqus/Standard*). These subroutines can be modified on restart; however, modifications should be made with caution because they might invalidate the solution from which the restart is being made.

Simultaneously Reading and Writing a Restart File

You can continue a previous analysis as a restart analysis and write the results from the restart analysis to a new restart file or state file. For example, if the previous analysis is an Abaqus/Explicit procedure and in the current analysis you specify that the restart point is Step *p* and the restart output frequency is *n*, the analysis will be restarted from the last saved interval of Step *p* and restart states will be written in subsequent steps based on the new value of *n*.

To discontinue the writing of a restart file in Abaqus/Standard when you are restarting a previous analysis, specify a restart output frequency of 0; if you do not specify a frequency, the file will continue to be written at the frequency defined for the previous analysis.

The New Restart File

Restart files can be very large for large models or for jobs involving many restart increments (unless you choose to overlay the restart data—see *Overlaying Results in the Restart Files*). Therefore, the previous restart file is not copied to the beginning of the new restart file when a job is restarted: only the data at restart increments requested in the current run are saved to the new restart file.

Example: Abaqus/Standard

Suppose an Abaqus/Standard job stopped running because it ran out of disk space. The last complete information for an increment in the restart file is from Step 2, increment 4. The following two-line input file could be used to restart this job and continue writing the restart file:

```
*HEADING
*RESTART, READ, STEP=2, INC=4, WRITE
```

Example: Abaqus/Explicit

Suppose you stopped an Abaqus/Explicit job because too much output was being generated. The last information in the state file is from Step 2, Interval 4 at a time of .004. Step 2 has a time period of .010 and restart results were requested at 10 intervals. The following input file could be used to restart this job and redefine the remainder of the step with reduced output requests:

```
*HEADING
*RESTART, READ, END STEP, STEP=2, INTERVAL=4
*STEP
*DYNAMIC, EXPLICIT
 , .006
*RESTART, WRITE, NUMBER INTERVAL=2
*END STEP
```

Continuation of Output on Restart

When you restart an analysis, Abaqus creates a new output database file (*job-name*`.odb`) and a new results file (*job-name*`.fil`) and writes output data to those files according to the criteria described below.

Output Database (`.odb`) Files

The Abaqus output database file (*job-name*`.odb`) contains results that can be used for postprocessing in Abaqus/CAE. By default, the output database file is not made continuous across restarts; Abaqus creates a new output database file each time a job is run. You can combine *X*–*Y* data extracted from multiple output database files in the Visualization module of Abaqus/CAE. Alternatively, you can also join field and history results from

an original analysis and a restart analysis by running the **abaqusrestartjoin** execution procedure. For more information, see *Joining Output Database (.odb) Files from Restarted Analyses*.

Results (`.fil`) Files

The Abaqus results file created in Abaqus/Standard and Abaqus/Explicit (*job-name*`.fil`) contains user-specified results that can be used for postprocessing in external postprocessing packages. In Abaqus/Explicit results are also written to the selected results file (*job-name*`.sel`), which is then converted to the results file for postprocessing. See *About Output* for details.

On restart Abaqus/Standard will copy the information from the old results file into the results file for the new job up to the restart point and begin writing the new results to the new file following that point. Abaqus/Explicit will copy the information from the old selected results file into the selected results file for the new job up to the restart point and begin writing the new results to the new file following that point.

If the old results file is not provided, Abaqus/Standard will continue the analysis, writing the results of the restart analysis only to the new results file. Therefore, you will have segments of the analysis results in different files, which should be avoided in most cases since postprocessing programs assume that the results are in a single continuous file. You can merge such segmented results files, if necessary, by using the **abaqusappend** execution procedure (*Joining Results (.fil) Files*).

Restart Compatibility

A restart analysis in Abaqus/Standard can use the restart files generated from the same maintenance delivery of the same general release. Restart might not be compatible between general releases and hotfix releases within the same general release. In Abaqus/Explicit the original analysis and the restart analysis must use precisely the same release.

A restart analysis in Abaqus and a recover analysis in Abaqus/Explicit must be run on a computer that is binary compatible with the computer used to generate the restart files. You should only use restart files generated from the same maintenance delivery of a general release for a restart analysis.

Importing and Transferring Results

In this section:

- *About Transferring Results between Abaqus Analyses*
- *Transferring Results between Abaqus/Explicit and Abaqus/Standard*
- *Transferring Results from One Abaqus/Standard Analysis to Another*
- *Transferring Results from One Abaqus/Explicit Analysis to Another*

About Transferring Results between Abaqus Analyses

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Transferring Results between Abaqus/Explicit and Abaqus/Standard*
- *Transferring Results from One Abaqus/Standard Analysis to Another*
- *Transferring Results from One Abaqus/Explicit Analysis to Another*
- **IMPORT*
- **IMPORT ELSET*
- **IMPORT NSET*
- **IMPORT SURFACE*
- **IMPORT CONTROLS*
- **INSTANCE*
- *Transferring results between Abaqus analyses*

Overview

Abaqus provides the capability to import a deformed mesh and its associated material state from Abaqus/Standard into Abaqus/Explicit and vice versa. This capability is particularly useful in manufacturing problems; for example, the entire sheet metal forming process (which requires an initial preloading, forming, and subsequent springback) can be analyzed. In this case the initial preloading can be simulated with Abaqus/Standard using a static procedure and the subsequent forming process can be simulated with Abaqus/Explicit. Finally, the springback analysis can be performed with Abaqus/Standard.

Abaqus also provides the capability to transfer desired results and model information from an Abaqus/Standard analysis to a new Abaqus/Standard analysis or from an Abaqus/Explicit analysis to a new Abaqus/Explicit analysis, where additional model definitions may be specified before the analysis is continued. For example, during an assembly process an analyst might first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Standard or Abaqus/Explicit analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Standard or Abaqus/Explicit analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

Finally, Abaqus provides the capability to transfer desired results and model information from multiple Abaqus/Standard analyses or multiple Abaqus/Explicit analyses to a new Abaqus/Explicit analysis, where additional model definitions can be specified before the analysis is continued. For example, during an assembly process an analyst might first be interested in the local behavior of a number of components but later is concerned with the behavior of the assembled product. In this case the local behavior of each component can first be analyzed individually in a series of Abaqus/Standard or Abaqus/Explicit analyses. Subsequently, the model information and results from these analyses can be transferred to a new Abaqus/Explicit analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

By default, imported model data such as nodes and elements, node sets and element sets, sections and materials, etc., are not required in the import analysis input because these quantities are reinstated from the restart database. However, you can choose to use a full-model input format in which all model data are given in the import analysis input. In addition to providing the complete model data used in the import analysis, this option provides additional advantages in specifying certain import features. For more information, see *Full-Model Input for Import*.

For this capability to work, the same maintenance release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible. You should only use the import capability within the same maintenance delivery of a general release.

Saving the Analysis Results

The restart files from the original analysis contain the analysis results that are transferred from Abaqus/Standard or Abaqus/Explicit. Obtaining restart files is described in more detail in *Writing Restart Files*; brief summaries are provided below. By default, Abaqus/Standard does not write any restart information and Abaqus/Explicit writes results at the beginning and end of each step.

Saving Results from Abaqus/Standard

If the results are to be imported from an Abaqus/Standard analysis, the results from the original Abaqus/Standard job must be written to the restart (`.res`), analysis database (`.mdl` and `.stt`), part (`.prt`), and output database (`.odb`) files.

You can specify the increments at which restart information will be written. Restart information is always written at the end of a step in addition to the requested increments whenever you request restart data in Abaqus/Standard.

Input File Usage: **RESTART*, WRITE, FREQUENCY=*n*

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *n* in the **Frequency** column for each step

Saving Results from Abaqus/Explicit

If the results are to be imported from an Abaqus/Explicit analysis, the results from the original Abaqus/Explicit job must be written to the state (`.abq`) file at the time when transfer of the state of the deformed body is required. The state (`.abq`), restart (`.res`), analysis database (`.mdl` and `.stt`), package (`.pac`), part (`.prt`), and output database (`.odb`) files will be used for importing the results from Abaqus/Explicit.

You can specify whether the results are to be written at the exact time dictated by the specified time interval, *n*, during a step of an Abaqus/Explicit analysis or at the increment ending after the time dictated by the specified time interval. Results are always written at the end of a step, so it is not necessary to request results at the exact time intervals if results will be read only from the end of a step.

Input File Usage: Use the following option to request results at the increments ending immediately after each time interval:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=NO

Use the following option to request results at the exact time intervals:

**RESTART*, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=YES

Abaqus/CAE Usage: Step module: **Output->Restart Requests**: enter *n* in the **Number Interval** column; click to check the **Time Marks** column for each step if you want the results written at the exact time intervals

Specifying the Transfer of Model Data and Results

The import capability is used to transfer model data and results from one analysis to another. The following sections describe how to specify the import request. You can import element sets from models that are not defined

as assemblies of part instances, or you can import part instances from models that are defined as assemblies of part instances. In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Specifying the Transfer of Model Data and Results for Models That Are Not Defined as Assemblies of Part Instances

You can import element sets from a previous analysis to specify the transfer of model data and results for models that are not defined as assemblies of part instances. This import capability is illustrated in *Springback of two-dimensional draw bending* and *Axisymmetric forming of a circular cup*.

Each element set to be imported must have been defined in the original analysis. You can import any element set, including nested element sets and those with overlapping elements. An imported element set can also be a subset of another imported element set. The elements in these sets as well as the element set definitions are imported. Even though an element may be included in multiple imported elements sets, each element is imported only once in the import analysis. You cannot use element sets that are internal to the original analysis.

Input File Usage: Use the following option to import element sets from a previous analysis:

**IMPORT*
list of element sets that are to be imported

For example, the following input imports the element set definitions for BLANK1 and BLANK2 in addition to the elements and element set definition for BLANK:

Original analysis

**SHELL SECTION*, MATERIAL=STEEL1, ELSET=BLANK1
.00082, 5
**SHELL SECTION*, MATERIAL=STEEL2, ELSET=BLANK2
.00082, 5
**ELSET*, ELSET=BLANK
BLANK1, BLANK2

Import analysis

**IMPORT*
BLANK

To prevent any ambiguity regarding element and node definitions, the **IMPORT* option must be specified before any options that define additional model data in the input file. However, this is not required if full-model input is used.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Repositioning Elements in the Model

You can reposition elements in the element sets from their original positions in the previous analysis to new positions in the import analysis. The new position is determined by a translation and/or rotation of the original position relative to the origin of the global coordinate system. The positioning data apply to all elements in the list of imported element sets. Element sets that require different positioning data need to be grouped separately during import.

Input File Usage: Use the following option to reposition imported element sets from a previous analysis:

**IMPORT*
list of element sets that are to be imported and relocated
positioning data

Multiple **IMPORT* options need to be defined for element sets that require different positioning data.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Importing Model Data and Results of All Supported Element Sections One Time by Default

You can easily import all element sets in the previous analysis with a section definition that is supported for import. This method provides a simple way to import all elements in a model once per previous analysis without the need to list the element sets to be imported explicitly.

You can also use this method when importing from multiple previous analyses. In this case, you will need to ensure that there are no duplicate entity names from all the analyses and that the element and node numbers are unique in the import model.

If you do not use the full-model input as described in *Full-Model Input for Import*, the names of the element sets imported will be the same by default as those used in the previous analysis. If you use the full-model input as described in *Full-Model Input for Import*, you must specify the names of the imported element sets.

You can include repositioning data as described in *Repositioning Elements in the Model* to relocate all elements imported using the default all-section input to the same new position in the import analysis.

Input File Usage: Use the following option to import all element sets with a section definition that is supported for import from a previous analysis:

**IMPORT*, LIBRARY=*old-job-1*
blank line
positioning data

Importing Model Data and Results of Element Sets Multiple Times

You can import model data and results of element sets from a previous analysis multiple times. You must define a new name and a new position for an element set that has been imported more than once. You specify the old element set name used in the previous analysis followed by the new element set name to be used in the import analysis. The old element set name must have been defined in the previous analysis. The use of internal sets is not supported.

New elements and nodes are generated with new element and node numbers for the renamed element sets. You specify element and node offsets; the new numbers are obtained by adding the offsets to the old numbers used in the previous analysis. It is your responsibility to select appropriate element and node offsets to preserve uniqueness of element and node numbering in the model.

The new position is determined as described in *Repositioning Elements in the Model*. To prevent multiple elements in the model from occupying identical positions, an old element set name must not appear more than once in the list of imported element sets for each import definition. Element sets that require renaming must be grouped separately during import from those that do not require renaming.

Input File Usage: Use the following option to rename and reposition element sets to be imported from a previous analysis:

**IMPORT*, RENAME, EOFFSET=*element-offset*, NOFFSET=*node-offset*
old_name_elem_set1, new_name_elem_set1
old_name_elem_set2, new_name_elem_set2
…
positioning data

Separate **IMPORT* options need to be defined for element sets that require renaming and/or employ different positioning data. All **IMPORT* options must have the same value assigned to the UPDATE and STATE parameters, as shown in the following example:

**IMPORT*, UPDATE=NO, STATE=YES
ASSEMBLY_QUADRANT_1_LOWER,
ASSEMBLY_QUADRANT_1_UPPER,
**IMPORT*, UPDATE=NO, STATE=YES, RENAME, EOFFSET=80, NOFFSET=97
ASSEMBLY_QUADRANT_1_LOWER, ASSEMBLY_QUADRANT_2_LOWER
ASSEMBLY_QUADRANT_1_UPPER, ASSEMBLY_QUADRANT_2_UPPER
0., 0., 0.,
0., 0., 0., 0., 0., 1., 90.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Importing Element Set, Node Set, and Surface Definitions One Time

All element set and node set definitions associated with the imported elements are imported by default. For models that are not defined as assemblies of part instances, you can also selectively import only specified element set or node set definitions. This capability provides a convenient way of selectively reusing the element or node sets defined in the original analysis. However, any members of such sets that do not belong to the imported elements are removed from the specified sets.

For example, suppose three element sets—`SHELL3D`, `MEMB`, and `ALL`—are defined in the original analysis. Element set `ALL` contains all of the elements in element sets `SHELL3D` and `MEMB`, as well as other elements. You choose to import only the element sets `SHELL3D` and `MEMB` (i.e., the elements in these sets as well as the element set definitions). In addition, you selectively import the element set definition `ALL` (but not the elements in this set). If element 100 belongs to element set `ALL` but not to either element set `SHELL3D` or element set `MEMB`, it will not be imported and will be removed from the list of elements belonging to element set `ALL`. The imported element set definitions are processed before any node or element definitions; therefore, even if element 100 is subsequently redefined in the import analysis, it will not belong to element set `ALL` (unless it is explicitly assigned to element set `ALL` in the import analysis).

Only node and element sets defined in the original or previous import analysis are available for importing. New sets defined during a restart run cannot be imported.

You can also selectively import surface definitions. Surfaces are not imported if they are defined with elements and nodes that are not imported. All surface definitions are imported automatically if you import surface definitions but do not specify the surfaces to import. For models that are not defined as assemblies of part instances, all surface definitions associated with the imported elements are imported by default in an Abaqus/Standard to Abaqus/Standard import analysis.

Input File Usage: Use any or all of the following options immediately following the **IMPORT* option to import selected element set, node set, or surface definitions:

**IMPORT ELSET*
**IMPORT NSET*
**IMPORT SURFACE*

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances. You cannot selectively import element and node set definitions in Abaqus/CAE. All element and node set definitions are imported automatically.

You cannot import surface definitions in Abaqus/CAE.

Importing Element Set, Node Set, and Surface Definitions Multiple Times

You can import element sets, node sets, and surfaces from a previous analysis multiple times. Element set, node set, and surface definitions associated with the imported elements are not imported by default, and you must define a new name for an element set, a node set, or a surface that has been imported more than once. You specify the old element set name, node set name, or surface name used in the previous analysis followed by the new element set name, node set name, or surface name to be used in the import analysis. The old name must have been defined in the previous analysis, and use of internal names is not supported.

New elements and nodes are generated with new element and node numbers for the renamed sets and surfaces using the offsets that you specified for import; the new numbers are obtained by adding the offsets to the old numbers used in the previous analysis.

Element sets, node sets, and surfaces that require renaming must be grouped separately during import from those that do not require renaming.

Input File Usage: Use the following options to rename element sets, node sets, and surfaces to be imported from a previous analysis:

**IMPORT ELSET*, RENAME
**IMPORT NSET*, RENAME
**IMPORT SURFACE*, RENAME

The **IMPORT ELSET*, **IMPORT NSET*, and **IMPORT SURFACE* options must immediately follow an **IMPORT* option with the RENAME, EOFFSET, and NOFFSET parameters specified.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

You cannot import surface definitions in Abaqus/CAE.

Importing Model Data and Results of Element Sets from Multiple Previous Analyses in an Abaqus/Explicit Analysis

You can import model data and results of element sets from multiple Abaqus/Standard analyses or multiple Abaqus/Explicit analyses to an Abaqus/Explicit analysis. However, you cannot import from a mix of Abaqus/Standard and Abaqus/Explicit analyses.

You must specify the name of each previous analysis in an import definition. When importing from multiple analyses, you cannot specify the previous analysis name using the **oldjob** option (see *Abaqus/Standard and Abaqus/Explicit Execution*).

New elements and nodes are generated with new element and node numbers for the renamed element sets. You specify element and node offsets; the new numbers are obtained by adding the offsets to the old numbers used in the previous analysis. It is your responsibility to select appropriate element and node offsets to preserve uniqueness of element and node numbering in the model.

The new position is determined as described in *Repositioning Elements in the Model*. To prevent multiple elements in the model from occupying identical positions, an old element set name must not appear more than once in the list of imported element sets for each import definition.

Element sets that require renaming must be grouped separately during import from those that do not require renaming.

Input File Usage: Use the following options to rename and reposition element sets to be imported from two previous analyses:

**IMPORT*, LIBRARY=*old-job-1*
old_name_elem_set1,
old_name_elem_set2,
**IMPORT*, RENAME, EOFFSET=*element-offset*, NOFFSET=*node-offset*, LIBRARY=*old-job-2*
old_name_elem_set3, new_name_elem_set3
old_name_elem_set4, new_name_elem_set4
…
positioning data

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Specifying the Transfer of Model Data and Results for Models That Are Defined as Assemblies of Part Instances

You can import part instances from a previous analysis to specify the transfer of model data and results for models that are defined as assemblies of part instances. If you import more than one part instance, all import parameters must be the same for each imported part instance. Each instance name that you specify must be the same as the instance name in the original analysis. Only sets that are defined within the imported instance will be imported. All surfaces defined within the imported instance are imported if you import surface definitions. Sets and surfaces defined at the assembly level must be redefined in the import analysis. New set and surface definitions can be added upon import. You cannot assign new sections, material orientations, normals, or beam orientations to the imported part instance.

Input File Usage: Use the following options to import a part instance from a previous analysis:

**INSTANCE*, INSTANCE=*instance-name*
Additional set and surface definitions (optional)
**IMPORT*
**IMPORT SURFACE*
**END INSTANCE*

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

You cannot import surface definitions in Abaqus/CAE.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select the instances to which the initial state should be assigned

Repositioning Part Instances in the Model

You can import a part instance and specify a new position for the part instance in the imported model. The new position is determined by a translation and/or rotation of the original position relative to the origin of the assembly (global) coordinate system.

Input File Usage: Use the following options to import and reposition a part instance from a previous analysis:

**INSTANCE*, INSTANCE=*instance-name*
positioning data
Additional set and surface definitions (optional)
**IMPORT*
**IMPORT SURFACE*
**END INSTANCE*

Abaqus/CAE Usage: Repositioning part instances in the model is not supported in Abaqus/CAE.

Importing Part Instances in the Model Multiple Times

You can import a part instance from a previous analysis more than once. In each instance, you must define a new name and a new position for a part instance that has been imported more than once. You specify the old name of the part instance in the previous analysis and a new name for the part instance.

The new position is determined by a translation and/or rotation of the original position relative to the origin of the assembly (global) coordinate system. Sets defined within the part instance will be imported and repositioned. Surfaces defined within the imported instance are imported if you import surface definitions. Sets and surfaces defined at the assembly level must be redefined in the import analysis. New set and surface definitions can be added upon import. You cannot assign new sections, material orientations, normals, or beam orientations to the imported part instance. If you import more than one part instance, the part instances must be from the same output database (`.odb`) file and all import parameters must be the same for each imported part instance.

Input File Usage: Use the following options to import a part instance more than once from a previous analysis:

**INSTANCE*, NAME=*new instance-name*, INSTANCE=*old instance-name*
positioning data
Additional set and surface definitions (optional)
**IMPORT*
**IMPORT SURFACE*
**END INSTANCE*

Abaqus/CAE Usage: Importing part instances in the model multiple times is not supported in Abaqus/CAE.

Importing Part Instances in the Model from Multiple Previous Analyses in an Abaqus/Explicit Analysis

For models defined as assemblies of part instances, you must specify the name of each previous analysis in an instance definition when importing from multiple previous analyses. In each instance you must define a new name and a new position for a part instance that has been imported more than once. You specify the old name of the part instance in the previous analysis and a new name for the part instance.

The new position is determined by a translation and/or rotation of the original position relative to the origin of the assembly (global) coordinate system. Sets defined within the part instance are imported and repositioned. Surfaces defined within the imported instance are imported if you import surface definitions. Sets and surfaces defined at the assembly level must be redefined in the import analysis. New set and surface definitions can be added upon import. You cannot assign new sections, material orientations, normals, or beam orientations to the imported part instance. If you import more than one part instance, all import parameters must be the same for each imported part instance.

Input File Usage: Use the following options to import part instances from two previous analyses:

```
*INSTANCE, NAME=Quarter_S1, INSTANCE=Quarter_1,
LIBRARY=cplate_assem_step1
0., 0., 0.,
0., 0., 0., 0., 0., 1., 90.
*IMPORT, UPDATE=NO, STATE=YES
*END INSTANCE
*INSTANCE, NAME=Quarter_S2, INSTANCE=Quarter_1,
LIBRARY=cplate_assem_step2
0., 0., 0.,
0., 0., 0., 0., 0., 1., 180.
*IMPORT, UPDATE=NO, STATE=YES
*END INSTANCE
```

Abaqus/CAE Usage: Importing part instances in the model from multiple previous analyses is not supported in Abaqus/CAE.

Identifying the Analysis from Which the Data Will Be Obtained When Importing from a Single Previous Analysis

You must specify the name of the job from which the model and results data will be obtained.

Input File Usage: For all models you can enter the following input on the command line:

abaqus job=*job-name* **oldjob=***oldjob-name*

If the **oldjob** parameter is omitted, Abaqus will prompt for the job name (see *Abaqus/Standard and Abaqus/Explicit Execution*) even if the current job is an Abaqus/Explicit analysis that uses the recover option to restart from the last available step and increment in the state file.

Alternatively, for models defined as element sets, you can use the following option:

**IMPORT*, LIBRARY=*oldjob-name*

and for models defined as assemblies of part instances, you can use the following option:

**INSTANCE*, LIBRARY=*oldjob-name*

If you input more than one **IMPORT* option or **INSTANCE* option, the *oldjob-name* specified by the LIBRARY parameter must be the same for each option.

If the job name is specified on the command line using the **oldjob** option, the command line specification will take precedence over the LIBRARY parameter.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: **Job name:** *output-database-name*

Identifying the Individual Analysis from Which the Data Will Be Obtained When Importing from Multiple Previous Analyses

You must specify the name of the analysis from which the model data and results will be obtained.

Input File Usage: For models defined as element sets, you must use the following option:

**IMPORT*, LIBRARY=*oldjob-name*

For models defined as assemblies of part instances, you must use the following option:

**INSTANCE*, LIBRARY=*oldjob-name*

When importing from multiple previous analyses, the individual job name must be given by the LIBRARY parameter. You should not use the **oldjob** option on the command line (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Abaqus/CAE Usage: Importing part instances in the model from multiple previous analyses is not supported in Abaqus/CAE.

Importing Model Data

Element property definitions of imported elements can be redefined only if the reference configuration is updated (see *Updating the Reference Configuration*) and the material state is not imported (see *Importing the Material State*). In this case the material orientation definitions (*Orientations*), hourglass stiffness but not hourglass control definitions, and transverse shear stiffness definitions (in the case of shell elements) of the imported elements can also be redefined.

For other reference configuration and material state combinations, the information required to define the section for each imported element will be imported from the original analysis. All material orientations will be transferred from the original analysis to the import analysis. Material orientations that are associated with imported elements cannot be redefined in the import analysis. However, orientation names that are not associated with any imported elements can be reused in the import analysis.

Transverse shear stiffness for imported shell elements cannot be redefined; the values will be transferred from the original analysis. Hourglass stiffness for the imported elements cannot be redefined in an Abaqus/Standard import analysis; the default values will be used. The section control definitions (kinematic formulation, order of accuracy in the element formulation, and hourglass control approach) to be used for imported elements cannot be redefined (see *Transferring Results between Abaqus/Explicit and Abaqus/Standard* for details).

Mass adjustment contributions (see *Mass Adjustment*) applied to an element set are always included when the element set is imported. There is no need to redefine these contributions in the import analysis unless different mass adjustment is required for the element set.

Nonstructural mass contributions (see *Nonstructural Mass Definition*) associated with an element set are not imported. These contributions need to be redefined in the import analysis if they are to be included in the model.

Only nodes associated with the imported elements are imported. New nodes can be defined in the import analysis.

Nodes or elements that use the same numbers as nodes or elements being imported can be defined provided that the reference configuration is updated, the material state is not imported, and the import is not done from an instance library. The new definitions will overwrite the imported definitions. If the reference configuration is not updated, new nodes or elements cannot use the imported node and element numbers irrespective of whether or not the material state is imported.

During results transfer from an Abaqus/Standard analysis to another Abaqus/Standard analysis or from an Abaqus/Explicit to another Abaqus/Explicit analysis, the coordinates of imported nodes can be modified from their imported values by respecifying the nodal definitions if the reference configuration is updated and the material state is not imported. This modification of the coordinates of imported nodes is not allowed during transfer of results from Abaqus/Explicit to Abaqus/Standard or vice versa.

Importing Model Data Defined by a Distribution

While transferring results from one Abaqus/Standard analysis to another Abaqus/Standard analysis, most element or material properties defined by a distribution (see *Distribution Definition*) are imported along with the elements. The only exceptions are spatially varying thicknesses and orientation angles defined on the layers of composite shells and solids; in this case Abaqus issues an error message during input file preprocessing.

While transferring results from an Abaqus/Explicit analysis to an Abaqus/Standard analysis, the only spatially varying element properties defined by a distribution that can be imported are shell thicknesses, shell offset, and section orientations for shell and solid elements. If any other element or material properties are defined with a distribution, Abaqus issues an error message during input file preprocessing.

While transferring results from an Abaqus/Standard analysis to an Abaqus/Explicit analysis or from an Abaqus/Explicit analysis to another Abaqus/Explicit analysis, the only spatially varying element properties defined by a distribution that can be imported are shell thicknesses, shell offset, section orientations for shell and solid elements, orientation angles defined for shell sections on the layers of composite shells, and section stiffness matrices specified directly for general shell sections. If any other element or material properties are defined with a distribution, Abaqus issues an error message during input file preprocessing. If an element set consists of elements whose properties are defined by one or more distributions, the element set cannot be imported more than once.

Section and material properties of imported elements can be redefined with distributions only if the reference configuration is updated (see *Updating the Reference Configuration*) and the material state is not imported (see *Importing the Material State*). In this case the material orientation definitions (*Orientations*), hourglass stiffness but not hourglass control definitions, and transverse shear stiffness definitions (in the case of shell elements) of the imported elements can also be redefined.

Importing Results from an Abaqus/Standard Analysis (Other than a Direct Cyclic Analysis)

If the results are imported from an Abaqus/Standard analysis, you can specify the step and increment in the restart file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: **IMPORT*, STEP=*step*, INCREMENT=*increment*

For models that are defined as assemblies of part instances, the **IMPORT* option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *increment*

Importing Results from an Abaqus/Standard Direct Cyclic Analysis

If the results are imported from a direct cyclic analysis, you can specify the step and iteration number in the restart file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: **IMPORT*, STEP=*step*, ITERATION=*iteration*

For models that are defined as assemblies of part instances, the **IMPORT* option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *iteration*

Importing Results from an Abaqus/Explicit Analysis

If the results are imported from an Abaqus/Explicit analysis, you can specify the step and interval in the state file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: **IMPORT*, STEP=*step*, INTERVAL=*interval*

For models that are defined as assemblies of part instances, the **IMPORT* option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *interval*

Updating the Reference Configuration

Once the current model configuration of an Abaqus analysis is imported into Abaqus/Explicit or Abaqus/Standard, the analysis can be continued with or without updating the reference configuration to be the imported configuration. If the reference configuration is not updated to be the imported configuration, the displacements and strains are reported as total values relative to the original reference configuration and will, hence, be continuous. If the reference configuration is updated to be the imported configuration, displacements and strains reported in the import analysis are the total values relative to the updated reference configuration. This choice is useful if results need to be displayed relative to the imported configuration, such as may be desirable in springback analysis. The reference configuration cannot be updated if the imported analysis is geometrically linear.

The choice of whether or not to update the reference configuration can influence strain-free nodal adjustments associated with contact initialization in Abaqus/Standard. Strain-free adjustments can be used to resolve penetrations or gaps that exist in the reference configuration in Abaqus/Standard, so prior displacements are not considered by the strain-free adjustment algorithm upon import if the reference configuration is not updated. Strain-free nodal adjustments in Abaqus/Explicit are based on the current configuration rather than the reference configuration, so these adjustments are not sensitive to whether the reference configuration is updated in Abaqus/Explicit. Further details on strain-free adjustments are provided in *Default Contact Initialization Method*, *Contact Initialization for General Contact in Abaqus/Standard*, *Contact Initialization for General Contact in Abaqus/Explicit*, and *Contact Initialization for Contact Pairs in Abaqus/Explicit*.

If connector elements are imported, the configuration can be updated provided that the state is not imported.

When hyperelastic materials are imported, the configuration must be updated if the state is not imported.

Input File Usage: Use the following option to specify that the reference configuration is to be updated to the imported configuration:

**IMPORT*, STEP=*step*, UPDATE=YES

Use the following option to specify that the reference configuration should not be updated to the imported configuration:

**IMPORT*, STEP=*step*, UPDATE=NO

For models that are defined as assemblies of part instances, the **IMPORT* option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: toggle **Update reference configuration** on or off

Importing the Material State

You can specify whether or not the associated material state should be imported. If you choose to import the material state, the following are imported:

- stresses;
- equivalent plastic strains;
- back stresses for the kinematic hardening models;
- user-defined state variables;
- damage-related state variables for the concrete damaged plasticity model;
- damage-related state-variables for traction-separation response with cohesive elements;
- damage-related state variables for ductile metals;
- damage-related state variables for fiber-reinforced composites;
- maximum deviatoric strain energy density during deformation history for Mullins effect;
- internal strains and stresses for viscoelastic material models;
- transformation strains and fraction of martensite for superelastic material models; and
- connector state variables such as plastic strains, frictional slip, and damage state.

Thus, the state is imported correctly for further analysis only for the following:

- linear elasticity,
- Mises plasticity (including the kinematic hardening models),
- extended Drucker-Prager plasticity,
- crushable foam plasticity,
- Mohr-Coulomb plasticity,
- critical state (clay) plasticity,
- cast iron plasticity,
- concrete damaged plasticity,
- Johnson-Cook plasticity,
- hyperelasticity (including Mullins effect),
- hyperfoam,
- viscoelasticity,
- superelasticity,
- traction-separation response with damage for cohesive elements,
- damage for ductile metals,
- damage for fiber-reinforced composites,
- connector behavior,
- materials defined in user subroutines *UMAT* and *VUMAT*, and
- materials defined using the parallel rheological framework for nonlinear viscoelastic-elastoplastic behavior.

For all other material models only stresses will be imported. No other state variables will be imported.

If the material behavior is defined in a user subroutine, you must ensure that the *UMAT* and *VUMAT* are consistent.

If connector elements are imported, the state can be imported provided that the configuration is not updated.

When hyperelastic materials are imported, the state must be imported if the configuration is not updated. In an import analysis where you specify that the reference configuration should not be updated and the material state should not be imported, the material state is imported for elements associated with hyperelastic materials and any initial conditions of the state specified for such elements are ignored.

Input File Usage: Use the following option to specify that the material state should be imported:

**IMPORT*, STATE=YES

Use the following option to specify that the material state should not be imported:

**IMPORT*, STATE=NO

For models that are defined as assemblies of part instances, the **IMPORT* option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances. Abaqus/CAE always imports the material state. If you want to import only the deformed mesh, you can import an orphan mesh from a selected step and increment of an output database; see *What kinds of files can be imported and exported from Abaqus/CAE?*.

Importing Rigid Bodies

A rigid body defined with an element set in the original analysis will be imported by default if all elements in the element set are imported; that is, if all of the rigid body elements belong to the imported element sets. The reference node of an imported rigid body is imported automatically, and you should not specify a new reference node for the imported rigid body. If the reference node of an imported rigid body is defined by a node set, the reference node set can be included in the imported node sets. If all elements belonging to a rigid body are imported multiple times, the rigid body and its reference node are imported the same number of times automatically. Multiple import of a rigid body with pin nodes or tie nodes assigned to the rigid body is not supported.

If a rigid body consists of a union of multiple element sets and these sets are imported multiple times, you must define a node set for the rigid body reference node. This node set must be imported to provide a unique node offset for the reference node.

A rigid body from an original analysis cannot be partially imported; that is, the full complement of rigid body elements must be imported. An assembly-level rigid body from the original analysis with parts and instances can be imported only if it refers to a reference node set defined at the part instance level in the original analysis. If the reference node is defined at the assembly level, the set and the rigid body cannot be imported.

Importing Fluid Cavities

You can only import fluid cavities bounded by a surface. When you import a fluid cavity by importing the element sets that constitute the bounding surface, you can import the nodal state at the fluid cavity reference node (representing the pressure inside the fluid cavity) if you redefine the fluid cavity and its reference node in the import analysis and you import the material state. In addition, you must either import the bounding surface of the fluid cavity or input it again with the same surface definition as in the previous analysis.

When you import a fluid cavity multiple times by importing multiple instances of the element sets that constitute the bounding surface, you can import the nodal state at the fluid cavity reference node for each instance if you redefine the fluid cavity and its reference node for each instance and you import the material state. In addition, you must either import the bounding surface of the fluid cavity in each instance or input it again with the same surface definition as in the previous analysis. You must assign different names to the different instances of the fluid cavity, reference node set, and bounding surface to preserve uniqueness of the names in the import analysis. You also need to offset the node number of the fluid cavity reference node in each instance using the same node offset specified for the imported element sets on the bounding surface.

You can import the state of the fluid cavity for all combinations of Abaqus/Standard and Abaqus/Explicit import analyses. You can also import the state of the fluid cavity using full-model input. However, you cannot import the state of a fluid cavity that is not bounded by a surface.

Defining Constraints upon Import

Most constraints (such as multi-point constraints and surface-based tie constraints) are not imported from the original analysis and must be redefined in the import analysis. However, the transfer of surface-based tie constraint data in an Abaqus/Standard to Abaqus/Standard import analysis for models that are not defined as assemblies of part instances is an exception. In this case, Abaqus/Standard automatically imports surface-based tie constraints, together with their surfaces. In all cases, using the reference configuration of the original analysis without update ensures these constraints are reproduced identically in the import analysis.

If a new constraint is defined in the import analysis, it is important to ensure that the constraint is not in violation either in the reference configuration or in the starting configuration of the import analysis. These two configurations are one and the same for newly introduced nodes. If a new constraint involves nodes of the original analysis, it is appropriate to update the reference configuration for the import analysis (for more information, see *Updating the Reference Configuration*).

In an Abaqus/Explicit import analysis, the original configuration is used for the formation of tie constraints if the following conditions are met:

- a tie constraint is defined previously for the main and secondary surfaces, and
- both surfaces are imported with unchanged definitions.

In all other cases, Abaqus/Explicit uses the deformed configuration. By default, Abaqus/Explicit uses the protocol regardless of whether you specify the original or the updated configuration as the reference configuration for import. You can override the default protocol by specifying the configuration to be used for tie constraint formation for all imported surfaces.

In an Abaqus/Standard analysis with adaptive meshing and acoustic-to-structure tie constraints, the structural as well as the acoustic nodes might move from their initial positions. When such acoustic and structure meshes are imported from Abaqus/Standard into Abaqus/Explicit without updating the reference configuration, the acoustic elements at the interface might appear distorted when viewed in the undeformed plot mode in the Visualization module of Abaqus/CAE. This distortion appears because the reference configuration for the acoustic nodes is updated automatically, while the configuration for the non-acoustic nodes is not. The deformed plot at time=0 displays the correct mesh.

Input File Usage: Use the following option in an Abaqus/Explicit import analysis to specify the original configuration for all tie constraint formation for the imported model:

**IMPORT CONTROLS*, TIE CONFIG=OLD

Use the following option to specify the update configuration for all tie constraint formation for the imported model:

**IMPORT CONTROLS*, TIE CONFIG=NEW

Specifying a Tolerance for Shell Normals in the Updated Configuration

When the imported configuration is updated upon import, the mesh discretization may not satisfy the mesh geometry checks imposed in Abaqus/Explicit or Abaqus/Standard to evaluate whether or not a mesh is reasonable. In the case of highly warped shell elements it is possible that the normal at the center of the element that is calculated from the midsurface interpolation may differ from the normal that is interpolated from the rotated normals at the nodes. If the difference exceeds the tolerance specified, the analysis will terminate. This suggests that a fine mesh may be required to model areas of high curvature change to achieve a successful analysis.

The unit normal computed from the midsurface interpolation, $\mathbf{n_1}$, and that predicted by the interpolation of the rotated normals at the nodes, $\mathbf{n_2}$, must satisfy the condition:

$$1 - f_{tol} \leq |\mathbf{n_1} \cdot \mathbf{n_2}|,$$

where you can specify the tolerance, f_{tol}. If you do not specify a tolerance value, a default value of $f_{tol} = 0.1$ is used.

Input File Usage: If you update the reference configuration to be the imported configuration, you can specify a tolerance for error checking on shell normals:

**IMPORT CONTROLS*, NORMAL TOL=f_{tol}

Abaqus/CAE Usage: The shell normal tolerance is not supported in Abaqus/CAE.

Full-Model Input for Import

You can specify the full-model input format for import to Abaqus/Standard or Abaqus/Explicit. In the full-model input format, all data related to the imported model are to be given in the input together with new model data

for any additional parts or components. In this way, the input provides complete information on the entire import analysis model.

You can mix the full-model input format with the default (non-full-model) input format in a chained import analysis sequence. You can use the full-model input format to import from an old import job that uses the default input format and vice-versa.

You cannot use the full-model input format with part instances. In general, limitations for import analyses using the default and the full-model input formats are the same.

Specifying the Full-Model Input Format

You specify the full-model input format once in an import analysis.

Input File Usage: Use the following option to specify the full-model input format for import:

**IMPORT CONTROLS*, MODEL=YES

If you use the default (non-full-model) input format, you must place **IMPORT* at the start of the model specification before all other keywords. With the full-model input format, however, you can place **IMPORT* keywords anywhere in the model data (in other words, anywhere before the step data). However, the mandatory **IMPORT CONTROLS* keyword must follow the **IMPORT* keywords.

Specifying the Old Job for Import

When you use the full-model input format, you must supply information about the old job.

Input File Usage: Use the following option to specify the previous analysis:

**IMPORT*, LIBRARY=*oldjob-name*

In this case, you should not use the **oldjob** option on the command line (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Import of Element Sets Multiple Times

You can import an element set multiple times from a previous analysis using the full-model input format. You will need to assign each instance of the element set a unique name. In addition, all elements and element nodes in the set should be assigned element numbers (ID) and node numbers that are properly offset, such that the element numbers and node numbers are unique in the entire import model. The imported element set names should be given as data for multiple import options, each with the proper values of element and node offsets.

Input File Usage: You can use the following option to specify element and node offset values and reposition data for element sets to be imported mulitple times from a previous analysis:

**IMPORT*, EOFFSET=*element-offset*, NOFFSET=*node-offset*
new_name_elset1, new_name_elset2, new_name_elset3, ...
positioning data

You must define multiple **IMPORT* options for element sets that require different offset and positioning data. All **IMPORT* options must have the same value assigned to the UPDATE and STATE parameters.

Use the following option to import the shell element set `ELSET_1` twice from the previous analysis. The set is imported the first time with the name `ELSET_1_1`, and with the same element and node numbers (no offsets). It is imported a second time with name `ELSET_1_2`, and with an element number offset of 100, a node number offset of 1000, and a repositioning translation of the nodes of 4.0 units along the Z-axis:

```
*NODE
1,         0.0,    0.0,    0.0
2,         1.0,    0.0,    0.0
3,         2.0,    0.0,    0.0
4,         0.0,    1.0,    0.0
5,         1.0,    1.0,    0.0
6,         2.0,    1.0,    0.0
1001,      0.0,    0.0,    4.0
1002,      1.0,    0.0,    4.0
1003,      2.0,    0.0,    4.0
1004,      0.0,    1.0,    4.0
1005,      1.0,    1.0,    4.0
1006,      2.0,    1.0,    4.0
*ELEMENT, TYPE=S4R
 11,       1,    2,    5,    4
 12,       2,    3,    6,    5
111,    1001,  1002,  1005,  1004
112,    1002,  1003,  1006,  1005
*ELSET, NAME=ELSET_1_1
11,  12
*ELSET, NAME=ELSET_1_2
111, 112
*IMPORT, UPDATE=NO, STATE=YES
ELSET_1_1
*IMPORT, UPDATE=NO, STATE=YES, EOFFSET=100, NOFFSET=1000
ELSET_1_2
0.,  0.,  4.
```

Nodal Coordinates of Imported Nodes

Although the data of all the imported nodes are given in the input using the full-model input format, the nodal coordinates to be employed in the analysis are reinstated from the restart database based on the repositioning data. Abaqus does not check for discrepancies between the coordinates of the reinstated nodes and those that are given in the full-model input.

Supported Results Formats for Import Analysis

When you use the full-model input format, you can specify the results format to be ODB or SIM when executing the import analysis.

Input File Usage: Use the following input on the command line to specifiy ODB results format:

abaqus resultsFormat=*ODB*

Use the following input on the command line to specifiy SIM results format:

abaqus resultsFormat=*SIM*

Use the following input on the command line to specifiy both ODB and SIM results format:

abaqus resultsFormat=*BOTH*

With the default (non-full-model) input format, you can only specify the ODB results format.

Partial Import of Elements of a Section

When you use full-model input, you can accomplish partial import of an element set by removing from the input data of the previous analysis elements and element sets that are not to be imported, leaving only those that are to be imported in the input. For example, suppose in a previous analysis a solid section is referenced by three solid element types: C3D4, C3D6, and C3D8R. If only the C3D8R elements are required in the import analysis, you can leave only the C3D8R elements in the section element set definition in the full model import. This provides added flexibility in managing the import workflow because no change is required with the previous analysis no matter what subset of elements you select for import.

Import of a Previously Rigid Element Set as Deformable

An element set that is defined as a rigid body in a previous analysis can be imported as deformable by importing the element set and removing its rigid body definition in the full-model input. In addition, the rigid body element set in the previous analysis must have its own section definition. In the import analysis, elements in the element set will have zero initial stress and the element nodes will acquire coordinates based on the deformed configuration imported from the previous analysis.

Limitations for Import from Multiple Previous Analyses

The capability to import from multiple previous analyses has the following known limitations:

- You can import only to an Abaqus/Explicit analysis.
- The previous analyses must all be Abaqus/Explicit analyses or Abaqus/Standard analyses. A mix of Abaqus/Explicit and Abaqus/Standard analyses is not supported.
- The use of either a part-instance model or a non-part-instance model but not both must be maintained in all previous analyses. A mix of previous analyses with and without instance definitions is not supported.
- All previous analyses must be imported with the same settings for updating the reference configuration and for importing the material state.
- All previous analysis models must have the same geometric dimension. A mix of two-dimensional, axisymmetric, and three-dimensional analyses is not supported.
- All limitations that pertain to import from a single Abaqus/Explicit analysis or a single Abaqus/Standard analysis remain effective.

Transferring Results between Abaqus/Explicit and Abaqus/Standard

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Transferring Results between Abaqus Analyses*
- **IMPORT*
- **IMPORT ELSET*
- **IMPORT NSET*
- **IMPORT SURFACE*
- **IMPORT CONTROLS*
- **INSTANCE*
- *Transferring results between Abaqus analyses*

Overview

Abaqus provides the capability to import a deformed mesh and its associated material state from Abaqus/Standard into Abaqus/Explicit and vice versa. In addition, new model information can be specified during the import analysis. This capability is useful for problems that involve several analysis stages. For example, in manufacturing processes the preloading can be analyzed using Abaqus/Standard and the subsequent forming operation can be simulated using Abaqus/Explicit. Finally, the springback of the material can be performed in Abaqus/Standard.

For this capability to work, the same release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in *About Transferring Results between Abaqus Analyses*.

Specifying New Data in an Import Analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New Model Definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see *Updating the Reference Configuration*). The usual Abaqus input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal Transformation

Nodal transformations (*Transformed Coordinate Systems*) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying Geometric Nonlinearity in an Import Analysis

By default, Abaqus/Standard uses a small-strain formulation (that is, geometric nonlinearity is ignored) and Abaqus/Explicit uses a large-deformation formulation (that is, geometric nonlinearity is included). For each step of an analysis, you can specify which formulation should be used; see *Geometric Nonlinearity* for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying Initial Conditions for Imported Elements and Nodes

Initial conditions (*Initial Conditions*) can be specified on the imported elements or nodes only under certain conditions. *Table 1* lists the initial conditions that are allowed depending on whether or not the material state is imported (see *Importing the Material State*). The reference configuration can be updated or not, as desired.

Table 1: Valid initial conditions.

Initial condition	Material state imported?
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Velocity	Yes or No
Void ratio	No

Procedures

Results can be imported into Abaqus/Explicit only from a general analysis step involving static stress analysis, dynamic stress analysis, or steady-state transport analysis in Abaqus/Standard. Results transfer from linear perturbation procedures (*General and Perturbation Procedures*) is not allowed.

Abaqus/Standard offers several analysis procedures that can be used in an import analysis. These procedures can be used to perform an eigenvalue analysis, static or dynamic stress analysis, buckling analysis, etc. See *Solving Analysis Problems* for a discussion of the available procedures.

For springback analysis of a formed component the first step in the Abaqus/Standard analysis usually consists of a static analysis procedure so that the initial out-of-balance forces can be removed gradually from the system. The removal of these forces is performed automatically by Abaqus/Standard during the first static analysis step, as described below. If the first step in the Abaqus/Standard analysis is not a static step (such as a dynamic step), the analysis proceeds directly from the state imported from the Abaqus/Explicit analysis.

Achieving Static Equilibrium When Importing into Abaqus/Standard

When the current state of a deformed body in an explicit dynamic analysis is imported into a static analysis, the model will not initially be in static equilibrium. Initial out-of-balance forces must be applied to the deformed body in dynamic equilibrium to achieve static equilibrium. Both dynamic forces (inertia and damping) and boundary interaction forces contribute to the initial out-of-balance forces. The boundary forces are the result of interactions from fixed boundary and contact conditions. Any changes in the boundary and contact conditions

from the Abaqus/Explicit analysis to the Abaqus/Standard analysis will contribute to the initial out-of-balance forces.

In general the instantaneous removal of the initial out-of-balance forces in a static analysis will lead to convergence problems. Hence, these forces need to be removed gradually until complete static equilibrium is achieved. During this process of removing the out-of-balance forces, the body will deform further and a redistribution of internal forces will occur, resulting in a new stress state. (This is essentially what occurs during "springback," when a formed product is removed from the worktools.)

When the first step in the Abaqus/Standard import analysis is a static procedure, the following algorithm is used to remove the initial out-of-balance forces automatically:

1. The imported stresses are defined at the start of the analysis as the initial stresses in the material.
2. An additional set of artificial stresses is defined at each material point. These stresses are equal in magnitude to the imported stresses but are of opposite sign. The sum of the material point stresses and these artificial stresses, thus, creates zero internal forces at the beginning of the step.
3. The internal artificial stresses are ramped off linearly in time during the first step. Thus, at the end of the step the artificial stresses have been removed completely and the remaining stresses in the material will be the residual stress state associated with static equilibrium.

Once static equilibrium has been obtained, subsequent steps can be defined using any analysis procedure that would normally follow a static analysis in Abaqus.

When the first step is not a static analysis, no artificial stress state is applied and the imported stresses are used in the internal force computations for the element.

Boundary Conditions

Boundary conditions, including any connector motion, specified in the original analysis are not imported. They must be defined again in the import analysis. In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see *Amplitude Curves*). If boundary conditions in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*), the same coordinate system should be defined and used in the import analysis.

For a discussion of applying boundary conditions, see *Boundary Conditions*.

Loads

Loads, including those applied for connector actuation, defined in the original analysis are not imported. Loads might, therefore, need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see *Amplitude Curves*). If point loads in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See *About Loads* for an overview of the loading types available in Abaqus.

Predefined Fields

Temperatures, whether they are prescribed or are degrees of freedom (as in a coupled thermal-stress analysis), and field variables at nodes are imported if the material state is imported.

If the reference configuration is updated and the material state is imported, the initial conditions for temperatures and field variables at the imported nodes are reset to the imported values; for example, the thermal strains are measured relative to the imported temperatures. If the reference configuration is updated but the material state is not imported, the initial conditions are reset to zero. In this case you can respecify the initial conditions on the imported nodes.

If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it is imported if the material state is imported.

Material Options

All material property definitions and the orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. All relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic Materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Connector Elements

When connector elements are imported, any associated connector behavior definitions are imported by default. The imported connector behavior definitions can be modified only if the state is not imported.

Mass Scaling

If mass scaling (*Mass Scaling*) is used in Abaqus/Explicit, the scaled masses will not be transferred to the subsequent import analysis in Abaqus/Standard. The mass of the model for the Abaqus/Standard analysis will be based on either the imported or the redefined density definitions.

Material Damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to Material Definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It might sometimes be possible to simplify the material definition. For example, if a Mises plasticity model was used in the Abaqus/Explicit analysis and no further plastic yielding is expected in the Abaqus/Standard analysis (as is generally the case for springback simulations), a linear elastic material can be used for the Abaqus/Standard analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for first-order continuum, modified triangular and tetrahedral elements, conventional shell, continuum shell, membrane, beam (both linear and quadratic), pipe (linear), truss, connector, rigid, and surface elements that are common to both Abaqus/Explicit and Abaqus/Standard, as defined in *Table 2*.

Table 2: Common element types that can be transferred between Abaqus/Explicit and Abaqus/Standard.

Common element types
CPS3, CPS3T, CPS4R, CPS4RT, CPS6M, CPS6MT
CPE3, CPE3T, CPE4R, CPE4RT, CPE6M, CPE6MT
CAX3, CAX3T, CAX4R, CAX4RT, CAX6M, CAX6MT
C3D4, C3D4T, C3D6, C3D6T, C3D8, C3D8I, C3D8R, C3D8T, C3D8RT, C3D10,C3D10T, C3D10M, C3D10MT
M3D3, M3D4, M3D4R
R2D2
R3D3, R3D4
RAX2
S4, S4R, S3R, S4RT, S3RT
SC8R, SC8RT, SC6R, SC6RT
SAX1
SFM3D3, SFM3D4R
T2D2
T3D2
B21, B22, PIPE21
B31, B32, PIPE31
CONN2D2[1], CONN3D2[1]
AC2D3, AC2D4R, AC2D4, ACIN2D2
AC3D4, AC3D6, AC3D8R, AC3D8, ACIN3D3, ACIN3D4
ACAX3, ACAX4R, ACAX4, ACINAX2
COH2D4, COHAX4, COH3D6, COH3D8
MASS, ROTARYI
[1]Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit; but not vice versa.

When S3R shell elements are imported from Abaqus/Explicit into Abaqus/Standard, they are converted into degenerated S4R elements automatically. However, when S3R shell elements are imported from Abaqus/Standard into Abaqus/Explicit, they remain S3R elements. When C3D6 and C3D6T solid elements are imported from Abaqus/Explicit into Abaqus/Standard, the results at the single point integration are applied to both integration points in Abaqus/Standard and the full integration is used automatically. However, when C3D6 and C3D6T solid elements are imported from Abaqus/Standard into Abaqus/Explicit, only the results at the first integration point are imported and are used in the reduced integration. When quadrilateral and hexahedral acoustic finite elements are imported between Abaqus/Explicit and Abaqus/Standard, they are converted to or from reduced-integration types, as required.

The following restrictions apply to the import capability:

- Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit but not vice versa. Further, if connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- Rebars defined using rebar layers (*Defining Reinforcement*) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique (*Embedded Elements*) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property (*Defining Rebar as an Element Property*) cannot be imported.
- Infinite elements and fluid elements cannot be imported.
- Rigid elements for which the thickness is interpolated from the nodes in an Abaqus/Explicit analysis will not be imported into Abaqus/Standard.
- A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported when all the elements used to define the rigid body are included in the element sets specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.
- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.
- Failed elements in Abaqus/Explicit will not be imported into Abaqus/Standard.
- Elements that are being removed or are inactive (see *Element and Contact Pair Removal and Reactivation*) in Abaqus/Standard will not be imported into Abaqus/Explicit.
- Rigid surfaces will not be imported.
- For mass or rotary inertia elements that have anisotropic values, import of the elements with repositioning that includes a rotation is not supported. In this case, the elements can only have isotropic values.

When importing results from an Abaqus/Standard analysis to an Abaqus/Explicit analysis, each element set specified can contain only compatible element types listed in *Table 3*. Element types from different cells are not compatible and cannot be combined in the same element set.

Table 3: Compatible element types.

ACINAX2, ACIN2D2, ACIN3D3, ACIN3D4
CPE4R, CPE3, AC2D3, AC2D4
CPS4R, CPS3
CAX4R, CAX3, ACAX3, ACAX4
AC3D4, AC3D6, AC3D8, C3D8, C3D8R, C3D4, C3D6
M3D4R, M3D3, M3D4
R3D3, R3D4
S4R, S3R, SC6R, SC8R, S4
SFM3D3, SFM3D4R
CAX6M, C3D10M
C3D8T, C3D4T, C3D6T
SC6RT, SC8RT, S4T, S4RT, S3T, S3RT
MASS, ROTARYI

Using Section Controls in an Import Analysis

When transferring results between Abaqus/Standard and Abaqus/Explicit, it is important that the hourglass forces are computed consistently. The enhanced hourglass control formulation (see *Enhanced Hourglass Control Approach in Abaqus/Standard and Abaqus/Explicit*) is recommended for computing hourglass forces in the original as well as all subsequent import analyses.

Once section controls have been defined in the original analysis, they cannot be modified in any subsequent Abaqus/Standard or Abaqus/Explicit analysis. Therefore, if section controls are to be used in any one analysis in a series of import analyses, they must be specified in the very first analysis. The section controls specified for an element set in the original analysis will be used for the elements belonging to that element set in all subsequent import analyses.

Section controls other than the hourglass control formulation have appropriate defaults depending on the type of analysis and, generally, do not need to be changed. Nondefault values can be chosen subject to certain restrictions.

In an Abaqus/Standard analysis only the average strain kinematic formulation and second-order accurate element formulation are available; other kinematic formulations, element formulations, or section controls that are relevant only in an Abaqus/Explicit analysis can be specified in the Abaqus/Standard analysis. Such controls will be ignored in the Abaqus/Standard analysis but retained for the subsequent Abaqus/Explicit import analysis.

If a kinematic formulation other than average strain is used for solid elements in the Abaqus/Explicit analysis, the differences in the kinematic formulations might lead to errors in Abaqus/Standard if the elements are distorted or undergo large rotations.

Using the first-order accurate element formulation (default) in Abaqus/Explicit and the second-order accurate element formulation (the only available formulation) in Abaqus/Standard is not expected to cause significant errors, since the time increment size in Abaqus/Explicit is inherently small. One exception to this is when the Abaqus/Explicit analysis involves components undergoing several revolutions, in which case it is recommended that the second-order accurate element formulation be used in Abaqus/Explicit.

Input File Usage: Use the following options in the original analysis:

**MEMBRANE SECTION*, CONTROLS=*name1*, ELSET=*elset1*
**SHELL SECTION*, CONTROLS=*name2*, ELSET=*elset2*
**SHELL GENERAL SECTION*, CONTROLS=*name3*, ELSET=*elset3*
**SOLID SECTION*, CONTROLS=*name4*, ELSET=*elset4*

Use options similar to the following one in the original analysis:

**SECTION CONTROLS*, NAME=*name1*

Abaqus/CAE Usage: Define section controls when you assign the element type in the original analysis:

Mesh module: **Mesh->Element Type**: **Element Controls**

Membrane and Shell Element Thickness Computation

The computations for membrane and shell element thicknesses are described below.

Shell Elements Defined Using a General Shell Section

For shells defined using a general shell section, the current thickness is computed based on the effective Poisson's ratio, which is 0.5 by default, in both Abaqus/Explicit and Abaqus/Standard.

Input File Usage: **SHELL GENERAL SECTION*, POISSON=

Abaqus/CAE Usage: Property module: homogeneous or composite shell section editor: **Section integration: Before analysis**: **Advanced**: **Section Poisson's ratio**

Shell Elements Defined Using Shell Sections Integrated during Analysis and Membrane Elements

For shells defined using shell sections integrated during analysis and for membranes in Abaqus/Standard, the current thickness is computed based on the effective Poisson's ratio, which is 0.5 by default. In Abaqus/Explicit, on the other hand, the computation of the thickness could be based either on the effective Poisson's ratio or the through-thickness strains, with the computation based on the through-thickness strains used by default.

If you do not specify a section Poisson's ratio for shell sections integrated during analysis or for membrane sections in an original Abaqus/Explicit or Abaqus/Standard analysis, the thickness computations in the original and all subsequent import analyses are carried out using the default methods. In other words, the thicknesses in all Abaqus/Standard analyses are computed using the default effective Poisson's ratio of 0.5, while the thicknesses in all Abaqus/Explicit analyses are computed using the through-thickness strains.

When the section Poisson's ratio is assigned a numerical value in an original Abaqus/Standard or Abaqus/Explicit analysis, the thickness computations in the original analysis and all subsequent import analyses are performed using the specified value for the effective Poisson's ratio.

Input File Usage: Use one of the following options:

**SHELL SECTION*, POISSON=
**SHELL SECTION*, POISSON=MATERIAL
**MEMBRANE SECTION*, POISSON=
**MEMBRANE SECTION*, POISSON=MATERIAL

Abaqus/CAE Usage: Property module:
Homogeneous or composite shell section editor: **Section integration: During analysis**: **Advanced**: **Section Poisson's ratio**
Membrane section editor: **Section Poisson's ratio**

Contact Angle Computation in SLIPRING-Type Connector Elements

The contact angle, α, made by the belt wrapping around node *b* (see *Complex Connections*) is computed automatically in Abaqus/Explicit, ignoring the value specified within the Abaqus/Standard analysis.

Constraints

Most types of kinematic constraints (including multi-point constraints and surface-based tie constraints) specified in the original analysis are not imported and must be defined again in the import analysis; however, embedded element constraints are imported by default. See *About Kinematic Constraints* for a discussion of the various types of kinematic constraints.

Similarly, surface-based kinematic and distributing coupling constraints specified in the original analysis are not imported and must be defined again in the import analysis. If transfer of material state is specified for import, the displacements and rotations of the coupling reference nodes are transferred in Abaqus/Standard to Abaqus/Explicit import analysis. This also applies to reference nodes that do not belong to any imported elements. By transferring nodal results, the constraint will be in initial equilibrium in the imported model. The transfer of nodal results is not supported in Abaqus/Explicit to Abaqus/Standard import analysis.

Interactions

Contact definitions specified in the original analysis and the contact state are not imported. Contact can be defined again in the import analysis by specifying the surfaces and contact pairs; however, you might not be able to use the exact contact definitions that were used in the original analysis because of differences in the contact capabilities between Abaqus/Standard and Abaqus/Explicit.

The contact constraint enforcement might be different in Abaqus/Standard and Abaqus/Explicit. Examples of potential causes for differences include:

- Abaqus/Standard typically uses a “pure main-secondary” approach, whereas Abaqus/Explicit typically uses a “balanced main-secondary” approach.
- Depending on the contact formulations used, Abaqus/Standard and Abaqus/Explicit sometimes treat shell thicknesses and midsurface offsets differently.

Thus, when the contact conditions are defined in the import analysis, the contact state that existed in the previous analysis might not be reproduced at the beginning of the import analysis. This could lead to a redistribution of stresses and an analysis that differs from what you desire. In some cases this problem can be mitigated by using nondefault options, such as ignoring shell thicknesses in the contact calculations, to match behaviors in Abaqus/Standard and Abaqus/Explicit.

The cohesive contact state can be imported from Abaqus/Standard to Abaqus/Explicit. Specifically, the cohesive bonding status and damage state are transferred, which allows Abaqus/Explicit to already have preexisting bonds and accumulated cohesive damage at the beginning of the analysis. Abaqus/Explicit recalculates the cohesive strain at the beginning of the analysis and reestablishes the cohesion points per constraint, which generally results in nonzero cohesive normal stresses but zero cohesive shear stresses.

For a detailed description of the contact capabilities in Abaqus and the differences in the contact capabilities between Abaqus/Standard and Abaqus/Explicit, see *About Contact Interactions*.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. The output variables available in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*. The output variables available in Abaqus/Explicit are listed in *Abaqus/Explicit Output Variable Identifiers*.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for *UMAT* and *VUMAT*. Similarly, for a connector behavior, the plastic relative displacement (CUP), kinematic hardening shift force (CALPHAF), overall damage (CDMG), damage initiation criteria (CDIF, CDIM, CDIP), friction accumulated slip (CASU), and connector status (CSLST, CFAILST) will be continuous.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration. Accelerations are recomputed at the start of an import analysis in Abaqus/Explicit and might be different from those obtained at the end of an Abaqus/Standard analysis. The differences in accelerations arise from the recalculation of the internal forces created by the imported stresses using the Abaqus/Explicit element formulation algorithms.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible.
- The capability is not available for fluid elements, infinite elements, and spring and dashpot elements. Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit but not vice versa. See the discussion on *Elements* earlier in this section for further details.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- Node sets that are generated from existing element sets (see *Node Definition*) must be defined in the original analysis.
- Contact pair definitions and general contact definitions are not imported. Analytical rigid surfaces are not imported.
- In an Abaqus/Explicit analysis, surfaces associated with contact pairs are defined specially for that role. They cannot be imported from a previous Abaqus/Standard analysis and must be redefined in the Abaqus/Explicit import analysis. Similarly, in an Abaqus/Explicit to Abaqus/Standard import analysis, surfaces associated with contact pairs in the Abaqus/Explicit analysis cannot be imported and reused.
- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, Mullins effect, hyperfoam, viscoelasticity, Mises plasticity (including the kinematic hardening models), extended Drucker-Prager plasticity, crushable foam plasticity, Mohr-Coulomb plasticity, critical state (clay) plasticity, cast iron plasticity, concrete damaged plasticity, damage for cohesive elements, damage for ductile metals, or damage for fiber-reinforced composites. See *Importing the Material State* for details.
- If the state is imported for connector elements with behavior defined, the plastic displacements, the frictional slip, and the damage state are imported and the connector forces are recomputed. Some of the connector output variables, such as CU, are also recomputed on import. The recomputed variables might differ slightly at the point of import due to precision and algorithmic differences between the two solvers across import. See *Importing the Material State* for details.
- Loads, boundary conditions, multi-point constraints, and equations are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint is not imported unless the reference node is part of another element definition that is imported.
- Fluid cavity definitions are not imported. In addition, the reference node of a fluid cavity is not imported unless the reference node is part of another element definition that is imported.
- Element and contact pair removal/reactivation (*Element and Contact Pair Removal and Reactivation*) cannot be used in the first step of an import analysis in Abaqus/Standard. It can be used in the subsequent steps.
- For an Abaqus/Standard to Abaqus/Explicit import analysis in which elements in the Abaqus/Standard analysis were removed and reactivated in multiple steps (*Element and Contact Pair Removal and Reactivation*) and all elements are active for transfer at the import step, some of the element states, such as strains, might not be transferred correctly.
- In a series of Abaqus/Standard and Abaqus/Explicit import analyses in the order Abaqus/Explicit(1) Abaqus/Standard(1) Abaqus/Explicit(2) Abaqus/Standard(2), if elements in an element set are removed in the Abaqus/Standard(1) analysis, the subsequent Abaqus/Standard(2) import analysis does not recognize that this element set was removed in a previous analysis and fails with an error message stating that the element

set is not found in the restart file. Such failures can be avoided by using flattened input files and requesting only the active element sets for import.

- Section controls must be defined in the original analysis if any of a series of import analyses uses nondefault element formulations since section controls cannot be changed in an import analysis. See the discussion on *Using Section Controls in an Import Analysis* earlier in this section.
- The symmetric model generation capability (*Symmetric Model Generation*) cannot be used in an import analysis in Abaqus/Standard.
- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- An Abaqus/Standard import analysis where the reference configuration is not updated is not allowed if the adaptive meshing capability (*About ALE Adaptive Meshing*) was used in the previous Abaqus/Explicit analysis.
- Mesh-independent spot welds (see *Mesh-Independent Fasteners*) are not imported. These constraints can be redefined in the import analysis and are formed using the reference configuration of the import model. If the reference configuration is updated, the redefined constraints might not match the old constraints exactly due to the differences in geometry. If new constraints are defined and the reference configuration of the import model is not updated, they might not initially be in compliance if the nodes involved in the constraint have nonzero displacements. This might cause numerical difficulty and potential exit of the import analysis. In this case it is recommended that you update the reference configuration on import.
- The first step after an import when the reference configuration is updated should not be used to generate a substructure.
- For beam structures that have acute curvatures and undergo large permanent changes in curvatures, slightly different equilibrated configurations will be seen when using import depending on whether or not the reference configuration is to be updated to the imported configuration (see *Updating the Reference Configuration*). This configuration difference is due to beam element formulation differences between Abaqus/Standard and Abaqus/Explicit.

Input File Template

Transferring Results between Abaqus/Explicit and Abaqus/Standard Using Models That Are Not Defined as Assemblies of Part Instances:

Abaqus/Explicit analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Standard analysis:

```
*HEADING
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
*IMPORT SURFACE
Data lines to specify surface definitions to be imported
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to redefine linear elasticity
*PLASTIC
Data lines to redefine Mises plasticity
 …
*BOUNDARY
Data lines to redefine boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring Results between Abaqus/Standard and Abaqus/Explicit Using Models That Are Not Defined as Assemblies of Part Instances:

Abaqus/Standard analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*STATIC
 …
*RESTART, WRITE, FREQUENCY=n
*END STEP
```

Abaqus/Explicit analysis:

```
*HEADING
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
*IMPORT SURFACE
```

```
Data lines to specify surface definitions to be imported
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to redefine linear elasticity
*PLASTIC
Data lines to redefine Mises plasticity
 …
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Transferring Results between Abaqus/Explicit and Abaqus/Standard Using Models Defined as Assemblies of Part Instances:

Abaqus/Explicit analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Standard analysis:

```
*HEADING
Part definitions (optional)
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
*IMPORT SURFACE
*END INSTANCE
```

```
Additional part instance definitions (optional)
Assembly level set and surface definitions
 …
*END ASSEMBLY
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring Results between Abaqus/Standard and Abaqus/Explicit Using Models Defined as Assemblies of Part Instances:

Abaqus/Standard analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*STATIC
 …
*RESTART, WRITE, FREQUENCY=n
*END STEP
```

Abaqus/Explicit analysis:

```
*HEADING
Part definitions (optional)
*ASSEMBLY, NAME=Assembly-1
```

```
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
*IMPORT SURFACE
*END INSTANCE
Additional part instance definitions (optional)
Assembly level set and surface definitions
*END ASSEMBLY
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to redefine linear elasticity
*PLASTIC
Data lines to redefine Mises plasticity
 …
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Transferring Results from One Abaqus/Standard Analysis to Another

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Transferring Results between Abaqus Analyses*
- **IMPORT*
- **IMPORT ELSET*
- **IMPORT NSET*
- **IMPORT CONTROLS*
- **INSTANCE*
- *Transferring results between Abaqus analyses*

Overview

Abaqus provides the capability to transfer desired results and model information from an Abaqus/Standard analysis to a new Abaqus/Standard analysis, where additional model definitions may be specified before the analysis is continued. For example, during an assembly process an analyst may first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Standard analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Standard analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

For this capability to work, the same release of Abaqus/Standard must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in *About Transferring Results between Abaqus Analyses*.

Comparison with the Restart Capability

Both the import and restart capabilities in Abaqus/Standard allow for the transfer of results and model information from one Abaqus/Standard analysis to another Abaqus/Standard analysis. However, the two capabilities have been designed for different applications.

The restart capability allows a completed Abaqus/Standard analysis to be restarted and continued. The entire model and results from the original analysis are transferred to the restart run, where additional analysis steps can be defined. Not much new model data can be specified in the restarted analysis; only model information such as new amplitude definitions, new node sets, and new element sets are allowed. Detailed information on the restart capability is given in *Restarting an Analysis*.

The import capability also allows a completed Abaqus/Standard analysis to be continued. In addition, this capability allows for the analysis to be continued with only desired components from the original analysis; the entire model need not be transferred. New model data—such as elements, nodes, surfaces, contact pairs, etc.—can be specified during the import analysis. During the import analysis it is possible to choose whether only model information from the previous analysis is to be transferred or if the results associated with that model also are to be transferred.

For situations where the goal is to continue the original analysis with no change to the model information, it is recommended that the restart capability be used. For situations where the model information requires changes, or for cases where you require control over the transfer of results, the import capability should be used.

Specifying New Data in an Import Analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New Model Definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see *Updating the Reference Configuration*). The usual Abaqus/Standard input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal Transformation

Nodal transformations (*Transformed Coordinate Systems*) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying Geometric Nonlinearity in an Import Analysis

By default, Abaqus/Standard uses a small-strain formulation (that is, geometric nonlinearity is ignored). For each step of an analysis you can specify whether or not geometric nonlinearity should be included; see *Geometric Nonlinearity* for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying Initial Conditions for Imported Elements and Nodes

Initial conditions can be specified on the imported elements or nodes only under certain conditions. *Table 1* lists the initial conditions that are allowed depending on whether or not the material state is imported (see *Importing the Material State*). The reference configuration can be updated or not, as desired, with one exception: for initial temperature or field variable conditions, the reference configuration must be updated.

Table 1: Valid initial conditions.

Initial condition	Material state imported?
Field variable	No
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Temperature	No
Velocity	Yes or No
Void ratio	No

Procedures

Results can be imported only from a general analysis step involving static stress analysis, dynamic stress analysis, steady-state transport analysis, coupled temperature-displacement analysis, or thermal-electrical-structural analysis in Abaqus/Standard. Results transfer from linear perturbation procedures (*General and Perturbation Procedures*) is not allowed.

Abaqus/Standard offers several analysis procedures that can be used in an import analysis. These procedures can be used to perform an eigenvalue analysis, static or dynamic stress analysis, buckling analysis, etc. See *Solving Analysis Problems* for a discussion of the available procedures.

When results are transferred from an Abaqus/Standard dynamic analysis to another Abaqus/Standard analysis where the first step is a static procedure, the initial out-of-balance forces must be removed gradually from the system. The removal of these forces is performed automatically by Abaqus/Standard during the first static analysis step, as described below. If the first step in the Abaqus/Standard analysis is not a static step (such as a dynamic step), the analysis proceeds directly from the state imported from the previous Abaqus/Standard analysis.

Achieving Static Equilibrium When Importing from a Dynamic Analysis to a Static Analysis

When the current state of a deformed body in a dynamic analysis is imported into a static analysis, the model will not initially be in static equilibrium. Initial out-of-balance forces must be applied to the deformed body in dynamic equilibrium to achieve static equilibrium. Both dynamic forces (inertia and damping) and boundary interaction forces contribute to the initial out-of-balance forces. The boundary forces are the result of interactions from fixed boundary and contact conditions. Any changes in the boundary and contact conditions will contribute to the initial out-of-balance forces.

In general, the instantaneous removal of the initial out-of-balance forces in a static analysis will lead to convergence problems. Hence, these forces need to be removed gradually until complete static equilibrium is achieved. During this process of removing the out-of-balance forces, the body will deform further and a redistribution of internal forces will occur, resulting in a new stress state. (This is essentially what occurs during “springback,” when a formed product is removed from the worktools.)

When the first step in the Abaqus/Standard import analysis is a static procedure, the following algorithm is used to remove the initial out-of-balance forces automatically:

1. The imported stresses are defined at the start of the analysis as the initial stresses in the material.
2. An additional set of artificial stresses is defined at each material point. These stresses are equal in magnitude to the imported stresses but are of opposite sign. The sum of the material point stresses and these artificial stresses, thus, creates zero internal forces at the beginning of the step.
3. The internal artificial stresses are ramped off linearly in time during the first step. Thus, at the end of the step the artificial stresses have been removed completely and the remaining stresses in the material will be the residual stress state associated with static equilibrium.

Once static equilibrium has been obtained, subsequent steps can be defined using any analysis procedure that would normally follow a static analysis.

When the first step is not a static analysis, no artificial stress state is applied and the imported stresses are used in the internal force computations for the element.

Boundary Conditions

Boundary conditions specified in the original analysis are not imported; they must be redefined in the import analysis.

In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a

constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see *Amplitude Curves*). If boundary conditions in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*), the same coordinate system should be defined and used in the import analysis.

For discussions on applying boundary conditions and multi-point constraints, see *Boundary Conditions* and *About Kinematic Constraints*.

Loads

Loads defined in the original analysis are not imported. Therefore, loads may need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see *Amplitude Curves*). If point loads in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See *About Loads* for an overview of the loading types available in Abaqus/Standard.

Predefined Fields

Temperatures, whether they are prescribed or are degrees of freedom (as in a coupled thermal-stress analysis), and field variables at nodes are imported if the material state is imported.

If the reference configuration is updated and the material state is imported, the initial conditions for temperatures and field variables at the imported nodes will be reset to the imported values; for example, the thermal strains will now be measured relative to the imported temperatures. If the reference configuration is updated but the material state is not imported, the initial conditions are reset to zero. In this case you can respecify the initial conditions on the imported nodes.

If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it will be imported if the material state is imported.

Material Options

All material property definitions and orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. In this case all relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic Materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Material Damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to Material Definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It may sometimes be possible to simplify the material definition. For example, if a Mises plasticity model was used in the first Abaqus/Standard analysis and no further plastic yielding is expected in a subsequent Abaqus/Standard analysis, a linear elastic material can be used for the Abaqus/Standard analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for thermal-electrical-structural elements and a subset of the stress/displacement and coupled temperature-displacement continuum, shell, membrane, truss, rigid, and surface elements available in Abaqus/Standard. The complete list of supported elements is provided in *Table 2*. If elements that are removed (see *Element and Contact Pair Removal and Reactivation*) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.

Table 2: Element types that can be transferred from one Abaqus/Standard analysis to another.

Element Type	Supported Elements
Plane strain continuum	CPE3, CPE3H, CPE3T, CPE4, CPE4H, CPE4HT, CPE4I, CPE4IH, CPE4R, CPE4RHT, CPE4RT, CPE4T
	CPE6, CPE6H, CPE6M, CPE6MH, CPE6MHT, CPE6MT, CPE8, CPE8H, CPE8HT, CPE8R, CPE8RH, CPE8RHT, CPE8RT, CPE8T
Plane stress continuum	CPS3, CPS3T, CPS4, CPS4I, CPS4R, CPS4T
	CPS6, CPS6M, CPS6MT, CPS8, CPS8R, CPS8RT, CPS8T
Three-dimensional continuum	C3D4, C3D4H, C3D4T, C3D5, C3D5H, C3D6, C3D6H, C3D6T, C3D8, C3D8H, C3D8HT, C3D8I, C3D8IH, C3D8R, C3D8RH, C3D8RHT, C3D8RT, C3D8S, C3D8HS, C3D8T, Q3D4, Q3D6, Q3D8, Q3D8H, Q3D8R, Q3D8RH
	C3D10, C3D10H, C3D10HS, C3D10M, C3D10MH, C3D10MHT, C3D10MT, C3D15, C3D15H, C3D15V, C3D15VH, C3D20, C3D20H, C3D20HT, C3D20R, C3D20RHT, C3D20RT, C3D20T, C3D27, C3D27H, C3D27RH, Q3D10M, Q3D10MH, Q3D20, Q3D20H, Q3D20R, Q3D20RH, CSS8
Axisymmetric continuum	CAX3, CAX3H, CAX3T, CAX4, CAX4H, CAX4HT, CAX4I, CAX4IH, CAX4R, CAX4RH, CAX4RHT, CAX4RT, CAX4T
	CAX6, CAX6M, CAX6MH, CAX6MHT, CAX6MT, CAX8, CAX8H, CAX8HT, CAX8R, CAX8RH, CAX8RHT, CAX8RT, CAX8T, QAX3, QAX4, QAX6M, QAX8
Membrane	M3D3, M3D4R
Two-dimensional rigid	R2D2
Three-dimensional rigid	R3D3, R3D4
Axisymmetric rigid	RAX2
Three-dimensional shell	S4R, S3R, S4RT, S3RT, S4T, S3T
Axisymmetric shell	SAX1
Continuum shell	SC6R, SC8R, SC6RT, SC8RT
Surface	SFM3D3, SFM3D4R
Two-dimensional truss	T2D2, T2D2T
Three-dimensional truss	T3D2, T3D2T

Element Type	Supported Elements
Cohesive	COH2D4, COHAX4, COH3D6, COH3D8
Inertial	MASS, ROTARYI

The following element types cannot be imported:

- Acoustic elements
- Axisymmetric-asymmetric continuum and shell elements
- Beam elements
- Connector elements
- Coupled thermal-electrical elements
- Diffusive heat transfer/mass diffusion elements and forced convection/diffusion elements
- Generalized plane strain elements
- Gasket elements
- Heat capacitance elements
- Infinite elements
- Piezoelectric elements
- Special-purpose elements
- Substructures
- User-defined elements

In addition, the following restrictions apply to the import capability:

- Rebars defined using rebar layers (*Defining Reinforcement*) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique (*Embedded Elements*) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property (*Defining Rebar as an Element Property*) cannot be imported.
- A rigid body containing both deformable and rigid elements cannot be imported. A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported when all the elements used to define the rigid body are included in the element sets specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.
- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.
- For mass or rotary inertia elements that have anisotropic values, import of the elements with repositioning that includes a rotation is not supported. In this case, the elements can only have isotropic values.

Constraints

Most types of kinematic constraints specified in the original analysis are not imported and must be defined again in the import analysis. However, by default Abaqus/Standard imports surface-based tie constraints together with the surfaces to be tied for models not defined as assemblies of part instances. See *About Kinematic Constraints* for a discussion of the various types of kinematic constraints.

Similarly, surface-based kinematic and distributing coupling constraints specified in the original analysis are not imported and must be defined again in the import analysis. If transfer of material state is specified for import,

the displacements and rotations of the coupling reference nodes and the constraining forces and moments of the coupling nodes are transferred in the import analysis. This also applies to nodes that do not belong to any imported elements. By transferring nodal results, the constraint will be in initial equilibrium in the imported model.

Interactions

The various aspects of most surface-based mechanical contact definitions (including the surface, contact pair, and contact property definitions) can be imported. Thermal interactions, electrical interactions, and pore fluid surface interactions cannot be imported. Certain types of mechanical contact aspects—pressure, penetration loads, cohesive behavior, and debonded surfaces—cannot be imported. The most commonly used mechanical contact aspects—pressure-overclosure behavior, frictional behavior, and damping—can be imported.

For models defined with element sets that are imported once, the ability to import element-based and node-based surfaces is determined by whether or not the underlying elements and nodes defining these surfaces are imported. If the underlying elements or nodes of a surface are not imported, that surface will not be imported. Rigid surface definitions are imported when the associated secondary surface is also imported. Contact pairs along with the associated surface interaction definitions are imported provided that all the secondary and main surfaces used in the original definition of the contact pair are also imported. Other contact-related features (such as surface interaction, surface smoothing, and clearance options) are also imported along with the contact pair definitions.

For models defined with element sets that are imported multiple times, you can control the import of the contact pair definitions by importing the main and secondary surfaces. When the main and secondary surfaces of a contact pair are imported with the same repositioning, the contact pair and its associated surface interaction definitions are imported automatically. Other surface-dependent features (such as surface smoothing and clearance options) are also generated along with the contact pair definitions. The contact state associated with the generated contact pairs is imported if the material state is imported.

For models defined with part instances that are imported once or multiple times, you can control the import of the contact pairs by redefining the main and secondary surfaces of the contact pairs in each imported instance. In addition, you must redefine the contact pairs and the associated surface interactions in the import analysis. You must also redefine other contact-related features (such as surface smoothing and clearance options) if required. Abaqus/Standard imports the contact state associated with the defined contact pairs if you import the material state.

Contact conditions modeled with contact elements will be ignored during the transfer process.

The contact state associated with a stress/displacement analysis is imported if the material state is imported. If the reference configuration is updated, the accumulated contact strains will be set to zero. The contact state associated with thermal, electrical, or pore fluid surface interactions is not imported. The contact state associated with a crack propagation analysis is not imported; initially bonded contact surface definitions are not transferred. If a contact pair was inactive in the step from which the import was done due to the use of contact pair removal (see *Removing and Reactivating Contact Pairs*), it must be deactivated again in the first step of the import analysis.

Additional contact information can be defined in the import analysis by specifying new surfaces, contact pairs, and interactions. New contact pair definitions can use the imported surface interaction definitions.

For a detailed description of the contact capabilities in Abaqus/Standard, refer to *About Contact Interactions*.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. Output requests in the original analysis are not transferred to the import analysis; output requests in the import analysis have to be respecified. The output variables available in Abaqus/Standard are listed in *Abaqus/Standard Output Variable Identifiers*.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for *UMAT*.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Standard must be run on computers that are binary compatible.
- The capability is not available for fluid elements; infinite elements; and spring, dashpot, and connector elements. See the discussion on *Elements* earlier in this section for further details.
- Element sets and part instances cannot be imported more than once nor can they be repositioned.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- The contact state associated with thermal, electrical, and pore fluid surface interactions is not imported; the contact state associated with crack propagation is not imported.
- General contact definitions are not imported.
- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, hyperfoam, viscoelasticity, Mises plasticity, and damage for cohesive elements. See *Importing the Material State* for details.
- Loads, boundary conditions, multi-point constraints, and equations are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint is not imported unless the reference node is part of another element definition that is imported.
- Fluid cavity definitions are not imported. In addition, the reference node of a fluid cavity is not imported unless the reference node is part of another element definition that is imported.
- When you import part instances individually from a previous analysis that was defined as an assembly of part instances, reference nodes associated with rigid body or coupling constraints defined on the imported instances will not be available in the import analysis for load or boundary condition application.
- Pre-tension section definitions are not imported; they have to be redefined in the import analysis.
- Table collection definitions are not imported; they have to be redefined in the import analysis.
- The capability is not available for elements with composite solid section definitions.
- If the elements that are removed in the original analysis (see *Element and Contact Pair Removal and Reactivation*) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.
- The symmetric model generation capability cannot be used in an import analysis in Abaqus/Standard.
- An original analysis in which the symmetric model generation capability is employed cannot be imported into a steady-state transport analysis.

- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- There may be a slight discontinuity during the transfer of state variables for analyses using fully integrated, first-order continuum elements if the elements are significantly deformed and the reference configuration is updated.
- Mesh-independent spot welds (see *Mesh-Independent Fasteners*) are not imported. However, the spot weld reference nodes are imported and can be used to redefine spot welds in the import analysis. The locations of the spot weld reference nodes and projection points are computed based upon the reference configuration of the import analysis. Therefore, if the deformed configuration of the imported model is significantly different from its reference configuration, it is recommended that the reference configuration be updated.
- If the value of the friction coefficient is changed from the value given in the model data of the original analysis, the changed value must be respecified in the first step of the import analysis (see *Changing Friction Properties during an Abaqus/Standard Analysis*).
- The capability is not available if adaptive meshing (see *ALE Adaptive Meshing and Remapping in Abaqus/Standard*) is used in the original analysis.
- Enriched features (see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*) are not imported.
- Restart files from the original analysis are used in the analysis preprocessor and in the Abaqus/Standard execution in the import analysis. When the import job is run in parallel on computer clusters by using MPI-based parallelization, these restart files are copied to each host machine. The original job restart files are not decomposed to match the import analysis parallel domain and may be large relative to the local disk space available on the host machines. You can minimize this file size by requesting restart output only for the increment from which import will occur.
- During import from one general dynamic step to another general dynamic step, reaction forces may experience jumps due to different time integrators or different time integrator parameter settings between the steps.

Input File Template

Transferring Results Using Models That Are Not Defined as Assemblies of Part Instances:

First Abaqus/Standard analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*RESTART, WRITE
*END STEP
```

Abaqus/Standard import analysis:

```
*HEADING
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
```

```
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
**
*** Optionally define additional model information
**
*BOUNDARY
Data lines to redefine boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring Results Using Models Defined as Assemblies of Part Instances:

First Abaqus/Standard analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*STATIC
 …
*RESTART, WRITE, FREQUENCY=n
*END STEP
```

Abaqus/Standard import analysis:

```
*HEADING
Part definitions (optional)
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
*END INSTANCE
Additional part instance definitions (optional)
Assembly level set and surface definitions
 …
*END ASSEMBLY
```

```
**
*** Optionally define additional model information
**
*BOUNDARY
Data lines to define boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring Results from One Abaqus/Explicit Analysis to Another

Products: Abaqus/Explicit Abaqus/CAE

References:

- *About Transferring Results between Abaqus Analyses*
- **IMPORT*
- **IMPORT ELSET*
- **IMPORT NSET*
- **IMPORT SURFACE*
- **IMPORT CONTROLS*
- **INSTANCE*
- *Transferring results between Abaqus analyses*

Overview

Abaqus provides the capability to transfer desired results and model information from an Abaqus/Explicit analysis to a new Abaqus/Explicit analysis, where additional model definitions might be specified before the analysis is continued. For example, during an assembly process an analyst might first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Explicit analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Explicit analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

For this capability to work, the same release of Abaqus/Explicit must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in *About Transferring Results between Abaqus Analyses*.

Comparison with the Restart Capability

Both the import and restart capabilities in Abaqus/Explicit allow for the transfer of results and model information from one Abaqus/Explicit analysis to another Abaqus/Explicit analysis. However, the two capabilities have been designed for different applications.

The restart capability allows a completed Abaqus/Explicit analysis to be restarted and continued. The entire model and results from the original analysis are transferred to the restart run, where additional analysis steps can be defined. Not much new model data can be specified in the restarted analysis; only model information such as new amplitude definitions, new node sets, and new element sets are allowed. Detailed information on the restart capability is given in *Restarting an Analysis*.

The import capability also allows a completed Abaqus/Explicit analysis to be continued. In addition, this capability allows for the analysis to be continued with only desired components from the original analysis; the entire model need not be transferred. New model data—such as elements, nodes, surfaces, contact pairs, etc.—can be specified during the import analysis. During the import analysis it is possible to choose whether only model information from the previous analysis is to be transferred or if the results associated with that model also are to be transferred.

For situations where the goal is to continue the original analysis with no change to the model information, it is recommended that the restart capability be used. For situations where the model information requires changes, or for cases where you require control over the transfer of results, the import capability should be used.

Specifying New Data in an Import Analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New Model Definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see *Updating the Reference Configuration*). The usual Abaqus/Explicit input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal Transformation

Nodal transformations (*Transformed Coordinate Systems*) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying Geometric Nonlinearity in an Import Analysis

By default, Abaqus/Explicit uses a large-strain formulation. For each step of an analysis you can specify whether or not geometric nonlinearity should be included; see *Geometric Nonlinearity* for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying Initial Conditions for Imported Elements and Nodes

Initial conditions can be specified on the imported elements or nodes only under certain conditions. *Table 1* lists the initial conditions that are allowed depending on whether or not the material state is imported (see *Importing the Material State*). The reference configuration can be updated or not, as desired, with one exception: for initial temperature or field variable conditions, the reference configuration must be updated.

Table 1: Valid initial conditions.

Initial condition	Material state imported
Field variable	No
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Temperature	No
Velocity	Yes or No
Void ratio	No

Boundary Conditions

Boundary conditions (including connector motion) specified in the original analysis are not imported. They must be redefined in the import analysis.

In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see *Amplitude Curves*). If boundary conditions in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*), the same coordinate system should be defined and used in the import analysis.

For discussions on applying boundary conditions and multi-point constraints, see *Boundary Conditions* and *About Kinematic Constraints*.

Loads

Loads, including those applied for connector actuation, defined in the original analysis are not imported. Therefore, loads might need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see *Prescribing Nondefault Amplitude Variations*) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see *Amplitude Curves*). If point loads in the original analysis are applied in a transformed coordinate system (see *Transformed Coordinate Systems*) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See *About Loads* for an overview of the loading types available in Abaqus/Explicit.

Predefined Fields

Temperatures, whether they are prescribed or are degrees of freedom (as in a coupled thermal-stress analysis), and field variables at nodes are imported if the material state is imported.

If the reference configuration is updated and the material state is imported, the initial conditions for temperatures and field variables at the imported nodes will be reset to the imported values; for example, the thermal strains will now be measured relative to the imported temperatures. If the reference configuration is updated but the material state is not imported, the initial conditions are reset to zero. In this case you can respecify the initial conditions on the imported nodes.

If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it will be imported if the material state is imported.

Material Options

All material property definitions and orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. In this case all relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic Materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Connector Elements

When connector elements are imported, any associated connector behavior definitions are imported by default. The imported connector behavior definitions can be modified only if the state is not imported.

Material Damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to Material Definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It might sometimes be possible to simplify the material definition. For example, if a Mises plasticity model was used in the first Abaqus/Explicit analysis and no further plastic yielding is expected in a subsequent Abaqus/Explicit analysis, a linear elastic material can be used for the subsequent Abaqus/Explicit analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for a subset of the stress/displacement and coupled temperature-displacement continuum, shell, membrane, truss, connector, rigid, and surface elements available in Abaqus/Explicit. The complete list of supported elements is provided in *Table 2*. If elements that are removed (see *Element and Contact Pair Removal and Reactivation*) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.

Table 2: Element types that can be transferred from one Abaqus/Explicit analysis to another.

Element Type	Supported Elements
Plane strain continuum	CPE3, CPE4R, CPE4RT, CPE6M, CPE6MT, CPE3T
Plane stress continuum	CPS3, CPS4R, CPS4RT, CPS6M, CPS6MT, CPS3T
Three-dimensional continuum	C3D4, C3D4H, C3D4T, C3D5, C3D6, C3D6T, C3D8R, C3D8RT, C3D10,C3D10T, C3D10M, C3D10MT, C3D8, C3D8T, C3D8I
Axisymmetric continuum	CAX3, CAX4R, CAX3T, CAX4RT, CAX6M, CAX6MT
Membrane	M3D3, M3D4M3D4R
Two-dimensional rigid	R2D2
Three-dimensional rigid	R3D3, R3D4
Axisymmetric rigid	RAX2
Three-dimensional shell	S4R, S3R, S3, S4, S4RS, S4RSW, S3RS, S3T, S3RT, S4T, S4RT
Continuum shell elements	SC6R, SC8R, SC6RT, SC8RT
Axisymmetric shell	SAX1
Surface	SFM3D3, SFM3D4R
Two-dimensional truss	T2D2

Element Type	Supported Elements
Three-dimensional truss	T3D2
Two-dimensional beam	B21, B22
Three-dimensional beam	B31, B32
Connector elements	CONN2D2, CONN3D2
Cohesive	COH2D4, COHAX4, COH3D6, COH3D8
Infinite elements	CINPS4, CINPE4, CINAX4, CIN3D8, ACIN2D2, ACIN3D3, ACINAX2
Acoustic elements	AC2D3, AC2D4R, AC3D4, AC3D6, ACAX3, ACAX4R, AC3D8R
Inertial elements	MASS, ROTARYI

The following element types cannot be imported:

- Heat capacitance elements
- Eulerian elements (EC3D8R and EC3D8RT)
- Particle elements (PC3D)

In addition, the following restrictions apply to the import capability:

- Rebars defined using rebar layers (*Defining Reinforcement*) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique (*Embedded Elements*) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property (*Defining Rebar as an Element Property*) cannot be imported.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported when all the elements used to define the rigid body are included in the element sets specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.
- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.
- For mass or rotary inertia elements that have anisotropic values, import of the elements with repositioning that includes a rotation is not supported. In this case, the elements can only have isotropic values.

Constraints

Kinematic constraints (including multi-point constraints and surface-based tie constraints) specified in the original analysis are not imported and must be defined again in the import analysis. See *About Kinematic Constraints* for a discussion of the various types of kinematic constraints.

Similarly, surface-based kinematic and distributing coupling constraints specified in the original analysis are not imported and must be defined again in the import analysis. If transfer of material state is specified for import, the displacements and rotations of the reference nodes are transferred in the import analysis. This also applies to reference nodes that do not belong to any imported elements. By transferring nodal results, the constraint will be in initial equilibrium in the imported model.

Interactions

For general contact, the contact state is imported if general contact is defined in the original and import analyses and if all underlying elements associated with a given contact constraint are imported without repositioning or are repositioned using identical translation and rotation.

For contact defined by contact pairs, contact definitions specified in the original analysis and the contact state are not imported. Contact can be defined again in the import analysis by specifying the surfaces and contact pairs.

Additional contact information can be defined in the import analysis by specifying new surfaces, contact pairs, and interactions.

For a detailed description of the contact capabilities in Abaqus/Explicit, refer to *About Contact Interactions*.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. Output requests in the original analysis are not transferred to the import analysis; output requests in the import analysis must be respecified. The output variables available in Abaqus/Explicit are listed in *Abaqus/Explicit Output Variable Identifiers*.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for *VUMAT*. Similarly, for a connector behavior, the plastic relative displacement (CUP), kinematic hardening shift force (CALPHAF), overall damage (CDMG), damage initiation criteria (CDIF, CDIM, CDIP), friction accumulated slip (CASU), and connector status (CSLST, CFAILST) will be continuous.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Explicit must be run on computers that are binary compatible.
- The capability is not available for spring and dashpot elements. See the discussion on *Elements* earlier in this section for further details.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- Embedded elements must be redefined if the host element set is imported more than once in an Abaqus/Explicit import analysis.
- The contact state for contact pairs is not imported. You must redefine surfaces defined for contact pairs in the import analysis if required.

- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, hyperfoam, viscoelasticity, Mises plasticity, and damage for cohesive elements. See *Importing the Material State* for details. For a connector behavior, the plastic displacements, the frictional slip, and the damage state are imported and the connector forces are recomputed. See *Importing the Material State* for details.
- Loads, boundary conditions, multi-point constraints, equations, and surface-based tie constraints are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint is not imported unless the reference node is part of another element definition that is imported.
- Fluid cavity definitions are not imported. In addition, the reference node of a fluid cavity is not imported unless the reference node is part of another element definition that is imported.
- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- Mesh-independent spot welds (see *Mesh-Independent Fasteners*) are not imported. These constraints can be redefined in the import analysis and are formed using the reference configuration of the import model. If the reference configuration is updated, the redefined constraints might not match the old constraints exactly due to the differences in geometry. If new constraints are defined and the reference configuration of the import model is not updated, they might not initially be in compliance if the nodes involved in the constraint have nonzero displacements. This might cause numerical difficulty and potential exit of the import analysis. In this case it is recommended that you update the reference configuration on import.

Input File Template

Transferring Results Using Models That Are Not Defined as Assemblies of Part Instances:

First Abaqus/Explicit analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Explicit import analysis:

```
*HEADING
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
*IMPORT SURFACE
```

```
Data lines to specify surface definitions to be imported
**
*** Optionally define additional model information
**
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Transferring Results Using Models Defined as Assemblies of Part Instances:

First Abaqus/Explicit analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Explicit import analysis:

```
*HEADING
Part definitions (optional)
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
*IMPORT SURFACE
*END INSTANCE
Additional part instance definitions (optional)
Assembly level set and surface definitions
 …
*END ASSEMBLY
**
*** Optionally define additional model information
**
```

```
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Modeling Abstractions

All Abaqus models involve certain abstractions. In addition to the traditional abstractions associated with the finite element method, you can include techniques in your model to obtain more cost-effective solutions.

In this section:

- *Substructures*
- *Submodeling*
- *Matrices*
- *Symmetric Model Generation, Results Transfer, and Analysis of Cyclic Symmetry Models*
- *Periodic Media Analysis*
- *Meshed Beam Cross-Sections*
- *Using the Extended Finite Element Method*
- *Reduced Basis Generation*

Substructures

In this section:

- *Using Substructures*
- *Generating Substructures*
- *Generating Frequency-Based Substructures*

Using Substructures

Products: Abaqus/Standard Abaqus/CAE

References:

- *Generating Substructures*
- *Generating Frequency-Based Substructures*
- **SLOAD*
- **SUBSTRUCTURE PATH*
- **SUBSTRUCTURE PROPERTY*
- **SUBSTRUCTURE DAMPING*
- **SUBSTRUCTURE DAMPING CONTROLS*
- **SUBSTRUCTURE MODAL DAMPING*

Overview

Substructures:

- allow a collection of elements to be grouped together and all but the retained degrees of freedom eliminated on the basis of linear response within the group;
- are used in the same manner as any of the standard element types in the Abaqus element library once created as described in *Generating Substructures*;
- can be used in stress/displacement and in coupled acoustic-structural analyses (however, frequency-based substructures are supported only in direct steady-state dynamic analyses);
- have linear response but allow for large translations and large rotations;
- are particularly useful in cases where identical pieces appear several times in a structure (such as the teeth of a gear) since a single substructure can be used repeatedly;
- can be translated, rotated with respect to the global system, and reflected in a plane when they are used;
- are connected to the rest of the model by the retained degrees of freedom at the retained nodes;
- might contain a set of internal load cases and boundary conditions that can be activated and scaled;
- can include dynamic effects by including retained eigenmodes; and
- appear to the rest of the model as a stiffness, optional mass, damping, and a set of scalable load vectors.

Substructures

Substructures are collections of elements from which the internal degrees of freedom have been eliminated. Retained nodes and degrees of freedom are those that are recognized externally at the usage level (when the substructure is used in an analysis), and they are defined during generation of the substructure. Factors that determine how many and which nodes and degrees of freedom should be retained are discussed below and in *Generating Substructures*.

A substructure can be considered as a special type of element (and is sometimes referred to as a superelement). The retained nodes of a substructure form its connectivity. Multiple instances of a substructure (superelement) can appear in a model.

Why Use Substructures?

There are a number of good reasons to use substructures.

Computational Advantages

- System matrices (stiffness, mass) are small as a result of substructuring. Subsequent to the creation of the substructure, only the retained degrees of freedom and the associated reduced stiffness (and mass) matrix are used in the analysis until it is necessary to recover the solution internal to the substructure.
- Efficiency is improved when the same substructure is used multiple times. The stiffness calculation and substructure reduction are done only once; however, the substructure itself can be used many times, resulting in a significant savings in computational effort.
- Substructuring can isolate possible changes outside substructures to save time during reanalysis. During the design process large portions of the structure will often remain unchanged; these portions can be isolated in a substructure to save the computational effort involved in forming the stiffness of that part of the structure.
- In a problem with local nonlinearities, such as a model that includes interfaces with possible separation or contact, the iterations to resolve these local nonlinearities can be made on a very much reduced number of degrees of freedom if the substructure capability is used to condense the model down to just those degrees of freedom involved in the local nonlinearity.

Organizational Advantages

- Substructuring provides a systematic approach to complex analyses. The design process often begins with independent analyses of naturally occurring substructures. Therefore, it is efficient to perform the final design analysis with the use of substructure data obtained during these independent analyses.
- Substructures provide a clean and simple way of sharing structural information. In large design projects large groups of engineers must often conduct analyses using the same structures.
- Many practical structures are so large and complex that a finite element model of the complete structure places excessive demands on available computational resources. Such a large linear problem can be solved by building the model, substructure by substructure, and stacking these level by level until the whole structure is complete and then recovering the displacements and stresses locally, as required.

Substructure Size

The retained nodal degrees of freedom and the generalized degrees of freedom associated with the substructure dynamic modes form a full set of the substructure degrees of freedom. The total number of substructure degrees of freedom is called the substructure size. Abaqus limits the substructure size to 16,384 for substructures used in Abaqus and to 46,340 for substructures generated in Abaqus and used outside of Abaqus, such as for flexible body dynamics workflows (see *Generating a Flexible Body*).

Valid Procedures

Substructures can be used without restriction in the following procedures:

- *Static Stress Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Direct-Solution Steady-State Dynamic Analysis*
- *Natural Frequency Extraction*
- *Complex Eigenvalue Extraction*
- *Mode-Based Steady-State Dynamic Analysis*

- *Transient Modal Dynamic Analysis*

Substructures can also be used in the following procedures, but recovery of eliminated degrees of freedom is not supported:

- *Response Spectrum Analysis*
- *Random Response Analysis*

Using Substructures in Static Analysis

Substructuring introduces no additional approximation in linear static structural analysis: the substructure is an exact representation of the linear, static behavior of its members. The principal drawback to the use of substructures in stress/displacement analyses is that a substructure's stiffness matrix is fully populated (no zero terms) and, therefore, might be very large if the substructure has a large number of retained degrees of freedom. This, in turn, might mean that the wavefront of the model within which substructures are used might be large, thus leading to long computer times to solve the equations.

This difficulty can often be avoided by choosing the substructure's boundaries carefully or by reusing several smaller substructures rather than a single larger substructure. In some cases it is possible to take advantage of the fact that Abaqus/Standard allows individual degrees of freedom to be retained, rather than the whole set of degrees of freedom at a node. For example, in contact problems without friction only the displacement component normal to the surface need be retained for the contact solution. Nodal transformations can be helpful in orienting the displacement components at surface nodes for this purpose (see *Transformed Coordinate Systems*).

In a static analysis involving a substructure containing acoustic elements, the results will differ from the results obtained in an equivalent static analysis without substructures. The acoustic-structural coupling is taken into account in the substructure (leading to hydrostatic contributions of the acoustic fluid), while the coupling is ignored in a static analysis without substructures.

Using Substructures in Dynamic Analysis

Substructures introduce approximations in dynamic analysis. The default approach to the dynamic representation of a substructure is to reduce its mass and damping matrix with the same transformation as is used for its stiffness matrix, which is known as “Guyan reduction.” This approach assumes that the response between the eliminated and retained degrees of freedom is correctly represented by the static modes only. This representation might not be accurate if dynamic modes within the substructure are important. The dynamic representation might be improved for Guyan reduction by retaining additional physical degrees of freedom that are not required to connect the substructure to the rest of the model. For example, if the substructure is a plate or a beam, some transverse displacements (and, perhaps, in-surface rotation components) might be included as retained degrees of freedom for this purpose. For more details regarding Guyan reduction, see *Substructuring and substructure analysis*.

“Dynamic mode addition” can be used as an alternative to Guyan reduction. This approach involves adding generalized degrees of freedom associated with the eigenmodes extracted for the substructure. This improves dynamic behavior, but it introduces the additional cost of extracting the eigenmodes for the constrained substructure. For more details regarding dynamic mode addition, see *Substructuring and substructure analysis*.

The reduction methods can be applied simultaneously to different substructures within the same structure. Definition of the reduced mass matrix is discussed further in *Generating Substructures*.

Using Substructures in Geometrically Nonlinear Stress/Displacement Analysis

Substructures might undergo large motions if geometric nonlinearities are considered in a particular stress/displacement analysis (see *About Static Stress Analysis Procedures*). Abaqus/Standard will account for the large rigid body rotations and translations of the substructure. However, the substructure is assumed to undergo small (linear elastic) deformations at all times during the geometrically nonlinear analysis. An equivalent rigid body rotation for each substructure is computed during each equilibrium iteration using the retained nodes

of the substructure. The substructure's mass, damping, stiffness matrix (including the retained eigenmodes), and force vectors are then rotated appropriately using the equivalent rigid body rotation. Appropriate (rotated) linear perturbation displacements (strain-inducing displacements relative to the rotating reference configuration) are used to compute the internal force associated with the substructure. Degrees of freedom at a node should not be retained selectively if the substructure is to be used in geometrically nonlinear analysis. Coupled acoustic-structural substructures should not be used in geometrically nonlinear analyses.

Comparison with Component Mode Synthesis

The component mode synthesis method has been developed to permit the structure to be subdivided into components (substructures), with most of the analysis being done on the smaller components to develop an approximate model for the entire structure.

The substructures in Abaqus/Standard are, in fact, a particular case of the component mode synthesis method extended to allow for large rotations and translations of the substructure (component) in the geometrically nonlinear analysis. The component mode synthesis method is based on the assumption that the small deformations of a substructure can be modeled using a collection of modes. The most frequently used modes in the literature are typically referred to as follows:

- constraint modes, which are static shapes obtained by giving each retained degree of freedom in the substructure a unit displacement while holding all other retained degrees of freedom fixed;
- fixed-interface normal modes, which are obtained by fixing the retained degrees of freedom and computing the eigenmodes of the substructure;
- free-interface normal modes, which are obtained by computing the eigenmodes of the substructure with free (not fixed) retained degrees of freedom; and
- mixed-interface normal modes, which are obtained by fixing a part of the retained degrees of freedom and computing the eigenmodes of the substructure.

The constraint modes are precisely the static modes (see *Substructuring and substructure analysis*) used by Abaqus/Standard. You include these modes in the substructure's representation by specifying the degrees of freedom that are to be retained (see *Defining the Retained Nodal Degrees of Freedom*). The fixed-interface, free-interface, or mixed-interface normal modes are the eigenmodes extracted in the eigenfrequency extraction step at the generation level, and these modes represent particular cases of substructure dynamic modes allowed in Abaqus (see *Defining the Generalized Degrees of Freedom*). You include the dynamic modes in the substructure's representation by selecting the eigenmodes to be used.

Including Substructures in a Model

When a substructure is used in a model, it is assigned an element number and defined by nodes just like any other element.

Use an element definition (*Element Definition*) with a substructure identifier to include substructures in the definition of another substructure (nested substructure) or in an analysis model.

In the element definition you define the substructure's element number at the usage level and assign node numbers to the substructure's retained nodes. You can define more than one substructure per element definition.

Once a substructure is introduced by an element definition, it is treated like any other element in the model, except that its response can be linear only (although it can be used as a part of a model that includes nonlinear effects, including large displacements).

Using substructures requires that the substructure database be available. All the files generated for the substructure (including the *name*`.sim` file and, optionally, the substructure model data file and the *name*`.prt`, *name*`.stt`, and *name*`.mdl` files) must be available if recovering substructure results is performed in the substructure usage analysis.

Abaqus distinguishes between different substructures by their unique substructure names. The substructure name is a significant part of the substructure element definition.

Two methods are available for including substructures in a model: use of the generic substructure element type SUBSTR and use of the Z*n*-type identifier. Although Abaqus supports both methods, the method using the generic substructure element type SUBSTR is preferred over the method using the Z*n*-type identification.

Preferred Method for Including Substructures in a Model

The preferred method for including substructures in a model is to associate all elements with the generic substructure element type SUBSTR.

If the substructure database files are located in the user directory, you can specify only the substructure name. If the substructure database files are located outside of the user directory, you must include the absolute or relative path to the file location.

Input File Usage: Use the following option to include one or more substructures in a model:

**ELEMENT*, TYPE=SUBSTR, FILE=*name*

Abaqus/CAE Usage: Specifying the generic substructure element type SUBSTR is not supported in Abaqus/CAE.

Alternative Method for Including Substructures in a Model

The alternative method for including substructures in a model is to associate all elements with the Z*n* type identification. In this method, the substructure name is formed from a prefix and a Z*n*-type identifier (for example, *name*_Z*n*).

Input File Usage: Use the following option to use the alternative method to include a substructure in a model:

**ELEMENT*, TYPE=Z*n*, FILE=*name*

Abaqus/CAE Usage: Use the following option to use the alternative method to include a substructure in a model:

All modules: **File->Import->Part**: **File Filter**: **Substructure**

Repeat the import process for each substructure that you want to include in the model.

Ordering of the Substructure Nodes on the Usage Level

The node numbers that are used when a substructure is created and the node numbers that are associated with the substructure when it is used are entirely independent. The ordering of the retained nodes when the substructure is used can be defined in two different ways:

1. The nodes can be provided in the same order that they were listed in the substructure definition. In this case you must prevent the sorting of the retained nodes when you specify the retained degrees of freedom (see *Preventing the Degrees of Freedom from Being Sorted*). Duplicate nodes are not combined if the retained nodes are not sorted. Therefore, if the same nodes are specified more than once in the list of retained degrees of freedom to retain different degrees of freedom, the corresponding nodes at the usage level must appear the same number of times.

2. The substructure nodes must be specified in the same order as the retained nodes sorted into ascending numerical order according to their numbers used within the substructure. This approach is the default when you specify the retained degrees of freedom.

In either case you must ensure that the nodes match up properly whenever a substructure is used.

Interpreting the Model Output in the Data File

Substructures included in the model can have nested substructures; these nested substructure can also have nested substructures as well, up to 20 levels. All of these substructures constitute the substructure tree of the model, which is printed to the data file with the relevant indentation. For each substructure element, the tree shows the substructure element's label, the full name of the associated substructure, and the suffix for the recovery output database name.

If model definition data are written to the data file (*Controlling the Amount of analysis input file processor Information Written to the Data File*), substructure instances are identified in the data (`.dat`) file by the substructure identifier followed by an F and two digits that indicate the substructure database number. The full name of the substructure database associated with this number is also contained in the model output.

Defining the Substructure's Properties

You associate a property definition with each substructure in the model. The property definition serves the following purposes:

- It defines any translation, rotation, and reflection of the substructure at the usage level.
- It allows a tolerance to be set to ensure that the coordinates of the usage level nodes match the coordinates of the nodes used to generate the substructure.
- It controls using various sources of substructure damping in the dynamic analysis at the usage level.
- It controls the use of frequency-based substructures in direct steady-state dynamic analyses (see *Direct-Solution Steady-State Dynamic Analysis*).
- It applies the substructure display geometric representation defined at generation.
- It activates the monitoring of the solution at monitor nodes within the substructure (see *Monitoring the Substructure Solution*).

Input File Usage: **SUBSTRUCTURE PROPERTY*, ELSET=*name*

Abaqus/CAE Usage: Use the following options to define translation and rotation of the substructure:

Assembly module: **Instance->Translate** or **Instance->Rotate**

Reflection of the substructure is not supported in Abaqus/CAE.

Use the following option to apply constraints that connect the retained nodes with the usage level nodes:

Interaction module: **Constraint->Create**

Applying the Substructure Display Geometrical Representation

You can apply the coarse substructure display geometric representation.

Input File Usage: Use the following option to apply the coarse substructure display representation defined at substructure generation.

*SUBSTRUCTURE PROPERTY, ELSET=*name*, DISPLAY=YES

Abaqus/CAE Usage: Applying the coarse substructure display representation is not supported in Abaqus/CAE.

Monitoring the Solution at the Substructure Nodes

You can monitor the solution at a set of substructure monitor nodes selected during substructure generation (see *Monitoring the Substructure Solution*) If you activative substructure monitoring, Abaqus/Standard adds the monitor nodes to the usage model automatically and prints the added node numbers in the data (`.dat`) file. Abaqus/Standard creates history output requests automatically for the active degrees of freedom of the monitor nodes (displacements, rotations, acoustic pressure, etc.). In dynamic analysis procedures Abaqus/Standard also adds history output requests for the velocity and acceleration at the monitor nodes.

Input File Usage: Use the following option to activate solution monitoring at selected nodes within substructures.

*SUBSTRUCTURE PROPERTY, ELSET=*name*, MONITOR=YES

Abaqus/CAE Usage: Activating the solution monitoring within substructures is not supported in Abaqus/CAE.

Translating, Rotating, and Reflecting a Substructure

Translation, rotation, and/or reflection (in that order) of a substructure can be specified in a substructure property definition.

Specify a translation by giving a translation vector. Specify a rotation by giving two points, a and b, defining a rotation axis plus a right-handed angular rotation around that axis. Specify a reflection by giving three non-colinear points in the reflection plane.

A translation does not affect the substructure's stiffness or mass: the principal reason to apply a translation is to enable the tolerance check on nodal coordinates as discussed later. Rotation and/or reflection of a substructure affect the substructure's stiffness and mass. The substructure load case definitions are rotated and/or reflected in the same way as the substructure's stiffness and mass; therefore, all loads within substructure load cases are applied in the local directions associated with the substructure when it was created.

For distributed loads (for example, pressure loading of a surface) this application is precisely what is desired. However, distributed body forces in coordinate directions (BX, BY, BZ) are applied in the substructure's local directions instead of in the global directions, which might not be what is needed. Similarly, distributed loadings that depend on position (for example, hydrostatic pressure or centrifugal loads) are based on the substructure's local coordinates and not on the substructure position during usage. Be careful to ensure that loading of a rotated or shifted substructure is correct for its usage.

Whenever a substructure is translated, rotated, and/or reflected, the degrees of freedom at any retained nodes are with respect to the coordinate directions at the usage level. Therefore, if all of the degrees of freedom of a node are not retained or if a two-dimensional substructure is used in a three-dimensional model with rotation out of the x–y plane, additional degrees of freedom might be activated due to rotation and/or reflection. Be careful to check the validity of the substructure usage in such cases.

Setting a Tolerance on the Substructure Nodes

One difficulty with using large substructures is ensuring that the retained nodes in the substructure are connected to the correct nodes on the usage level (after substructure translation, rotation, and/or reflection, if applicable).

Therefore, Abaqus/Standard checks that the coordinates of the retained nodes match the coordinates of the corresponding nodes on the usage level. A substructure does not require any coordinates on the usage level because it consists only of a stiffness matrix, a mass matrix, and a number of load cases. Nevertheless, it is usually a good check of a model's validity to verify that the substructure and the model into which it is introduced are geometrically consistent.

To check the coordinates, you can set a tolerance on the distance between usage level nodes and the corresponding substructure nodes. This tolerance indicates the largest deviation allowable before a warning is issued. If you do not specify this tolerance, the default is to use a tolerance of 10^{-4} times the largest overall dimension within the substructure. If you specify a tolerance of 0.0, the position of the retained nodes is not checked.

The geometric check is based on the coordinates of the retained nodes after translation, rotation, and/or reflection of the substructure at the usage level; motions of these nodes that occur as a result of geometrically nonlinear preloading during generation of the substructure are not considered in this check.

Input File Usage: **SUBSTRUCTURE PROPERTY*, ELSET=*name*, POSITION TOL=*tolerance*

Abaqus/CAE Usage: Assembly module: **Instance->Translate** and **Instance->Rotate**

Defining Substructure Damping

Defining substructure damping at the substructure usage level means defining viscous and structural damping matrices for the finite elements associated with the substructures. Abaqus allows you to choose a particular source of damping for a substructure, to add several sources, or to exclude the damping effects for a substructure at the usage level. All options defining the substructure damping belong to a substructure property definition and affect only the finite elements of the substructure type associated with the substructure property.

Sources of Substructure Damping

You can choose to model the damping of a substructure at the usage stage by using the reduced substructure damping matrices computed during the generation stage and stored on the substructure database. We denote the reduced viscous damping matrix of a substructure as $\overline{C}_{reduced}$ and the reduced structural damping matrix of a substructure as $\overline{S}_{reduced}$. Alternatively, you can introduce the stiffness and mass proportional damping matrices by multiplying the reduced substructure stiffness and mass matrices, $\overline{K}_{reduced}$ and $\overline{M}_{reduced}$, respectively, with the factors defined within the substructure property definition at the usage stage. You can also combine both damping sources or exclude the effects of damping altogether at the usage level. Finally, you can introduce viscous and structural modal damping matrices for a substructure specifying damping coefficients for the substructure eigenmodes calculated at the generation stage and stored on the substructure database.

The substructure modal damping contributes to the damping matrices for the finite elements associated with a substructure, and it can be used instead of or together with the other substructure damping sources. To define the substructure modal damping matrix, you specify the diagonal damping matrix on the substructure modal subspace. This matrix is transformed to the substructure degrees of freedom space to be added to the damping matrix of the finite element associated with the substructure.

Controlling the Sources of Substructure Viscous Damping

In the general case the substructure type element viscous damping matrix at the usage stage is defined by the following matrix equation:

$$\overline{C} = \overline{C}_{reduced} + \alpha\overline{M} + \beta\overline{K} + \overline{C}_{modal}.$$

You can specify substructure viscous damping using substructure damping controls and/or substructure viscous modal damping. If you specify substructure viscous modal damping, it is used in combination with all other

activated viscous damping sources to form the viscous damping matrix of the finite element. Defining the substructure viscous modal damping is discussed in more detail in *Defining Substructure Viscous Modal Damping* below.

Input File Usage: To activate only the generated condensed viscous damping matrix of the substructure (the first term on the right-hand side), use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, VISCOUS=ELEMENT

To activate only the stiffness and mass proportional substructure viscous damping, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, VISCOUS=FACTOR

To activate the combined generated and proportional substructure viscous damping matrix, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, VISCOUS=COMBINED

To exclude the effects of generated and proportional substructure viscous damping altogether at the usage level, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, VISCOUS=NONE

To specify the substructure viscous modal damping matrix (the last term on the right-hand side), use the following option:

**SUBSTRUCTURE MODAL DAMPING*

Abaqus/CAE Usage: Substructure damping controls and substructure modal damping are not supported in Abaqus/CAE.

Controlling the Sources of Substructure Structural Damping

In the general case the substructure type element structural damping matrix is defined by the following equation:

$$\overline{S} = \overline{S}_{reduced} + \gamma \overline{K}_{reduced} + \overline{S}_{modal}.$$

You can specify substructure structural damping using substructure damping controls and/or substructure structural modal damping. If you specify substructure structural modal damping, it is used in combination with all other activated structural damping sources to form the structural damping matrix of the finite element. Defining the substructure structural modal damping is discussed in more detail in *Defining Substructure Structural Modal Damping* below.

Input File Usage: To activate only the generated reduced structural damping matrix of the substructure (the first term on the right-hand side), use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, STRUCTURAL=ELEMENT

To activate only the stiffness proportional substructure structural damping matrix, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, STRUCTURAL=FACTOR

To activate the combined generated and stiffness proportional structural damping matrix, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, STRUCTURAL=COMBINED

To exclude the generated and stiffness proportional structural damping matrices, use the following option:

**SUBSTRUCTURE DAMPING CONTROLS*, STRUCTURAL=NONE

To specify the substructure structural modal damping matrix (the last term on the right-hand side), use the following option:

**SUBSTRUCTURE MODAL DAMPING*

Abaqus/CAE Usage: Substructure damping controls and substructure modal damping are not supported in Abaqus/CAE.

Defining Substructure Damping Factors

By default, the damping factors, α and β, and the structural damping factor, γ, used to define stiffness proportional and mass proportional damping for a substructure are zeros.

Input File Usage: Use the following options to define the values of the substructure damping factors at the usage level:

**SUBSTRUCTURE DAMPING*, ALPHA=α, BETA=β, STRUCTURAL=γ

Abaqus/CAE Usage: Defining substructure damping factors is not supported in Abaqus/CAE.

Defining Substructure Viscous Modal Damping

Substructure viscous modal damping is defined for the substructure eigenmodes extracted at the substructure generation level. The mode numbers and the eigenfrequencies used to define substructure viscous modal damping come from the solution of the substructure eigenvalue problem at the generation level.

Input File Usage: Use the following option to define the fraction of critical damping for a substructure by specifying mode numbers:

**SUBSTRUCTURE MODAL DAMPING*, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define the fraction of critical damping for a substructure by specifying a frequency range:

**SUBSTRUCTURE MODAL DAMPING*, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE

Use the following option to define substructure modal Rayleigh damping by specifying the substructure mode numbers:

**SUBSTRUCTURE MODAL DAMPING*, VISCOUS=RAYLEIGH, DEFINITION=MODE NUMBERS

Use the following option to define substructure modal Rayleigh damping by specifying a frequency range:

**SUBSTRUCTURE MODAL DAMPING*, VISCOUS=RAYLEIGH, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Substructure modal damping is not supported in Abaqus/CAE.

Defining Substructure Structural Modal Damping

Substructure structural modal damping is defined for the substructure eigenmodes extracted at the substructure generation level. The mode numbers and the eigenfrequencies used to define substructure structural modal damping come from the solution of the substructure eigenvalue problem at the generation level.

Input File Usage: Use the following option to define substructure structural modal damping by specifying mode numbers:

**SUBSTRUCTURE MODAL DAMPING*, STRUCTURAL, DEFINITION=MODE NUMBERS

Use the following option to define substructure structural modal damping by specifying a frequency range:

**SUBSTRUCTURE MODAL DAMPING*, STRUCTURAL, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Substructure modal damping is not supported in Abaqus/CAE.

Controlling Large Rotations of Substructures in Geometrically Nonlinear Analyses

Displacements within a substructure are assumed to correspond to rigid body motion plus small, linear-elastic deformation. Unrealistic results can occur for a simulation which involves large deformation of a submodel. When large strain effects are significant for the analysis, it is possible to get incorrect reaction forces, the deformed shape of the substructure might be inaccurate, or it might be difficult to achieve convergence. If a substructure with boundary conditions or if large rotations of a substructure are unlikely, you can improve the convergence and solution quality by suppressing the large rotations.

For details, refer to "Using Substructures in Nonlinear Abaqus/Standard Analyses" in the Dassault Systèmes Knowledge Base at *https://support.3ds.com/knowledge-base/*.

Input File Usage: Use the following option to suppress large rotations of the substructure in geometrically nonlinear analyses:

**SUBSTRUCTURE PROPERTY*, ELSET=*name*, LARGE ROTATIONS=NO

Abaqus/CAE Usage: Controlling large rotations of the substructure is not supported in Abaqus/CAE.

Controlling the Use of Frequency-Based Substructures

Frequency-based substructures are supported only in direct steady-state dynamic analyses (see *Direct-Solution Steady-State Dynamic Analysis*). When used at the same frequencies as those used for generation, frequency-based substructures represent the substructure with dynamic exactness. This allows for large models to reduce to very

small models without any loss of accuracy. By default, the frequency-based substructure is used only at those frequencies at which the frequency-based substructure is generated (see *Generating Frequency-Based Substructures*), and the conventional substructure is used at all the nonmatching frequencies. Therefore, the dynamic representation of the substructure is exact at the matching frequencies and is the same as that of the conventional substructure at nonmatching frequencies. The frequency-based substructure operators from the *jobname*_Z*n*.sim file obtained from the substructure generation procedure are used.

You can choose to use the frequency-based substructure at all frequencies of a direct steady-state dynamic analysis. However, the response at frequencies that do not match those used at generation is only an approximation. This approximation is based on the averaging of the frequency-based substructure operators corresponding to neighboring frequencies. If the frequency of interest is beyond the range of frequencies at which the frequency-based substructure is generated, this averaging is not possible, and Abaqus issues an error in the *jobname*.msg file. You can also disable the use of frequency-based substructures at any frequency, even if the frequency-based substructure operators are available in the *jobname*_Z*n*.sim file. In this case the conventional substructure is used instead at all frequencies.

Substructure modal damping is allowed only when the substructure does not contain frequency-based substructures or the use of frequency-based substructures is disabled; otherwise, substructure modal damping is ignored. Similarly, substructure loads are allowed in the direct steady-state dynamic analysis only when the substructure does not contain frequency-based substructures or the use of frequency-based substructures is disabled.

Input File Usage: Use the following option to use the frequency-based substructure when frequencies of interest match those frequencies at which the frequency-based substructure is generated and to use the conventional substructure at other frequencies:

**SUBSTRUCTURE PROPERTY*, ELSET=*name*, FREQUENCY BASED=MATCHED FREQUENCIES (default)

Use the following option to use the frequency-based substructure at all frequencies:

**SUBSTRUCTURE PROPERTY*, ELSET=*name*, FREQUENCY BASED=ALL FREQUENCIES

Use the following option to disable the use of the frequency-based substructure at all frequencies and to use the conventional substructure at all frequencies instead:

**SUBSTRUCTURE PROPERTY*, ELSET=*name*, FREQUENCY BASED=NO FREQUENCIES

Abaqus/CAE Usage: Frequency-based substructures are not supported in Abaqus/CAE.

Defining Kinematic Constraints and Transformations

All kinematic boundary conditions, MPCs, and transformations can be applied to retained degrees of freedom at the usage level. These specifications can be changed from step to step in the usual way. In this respect substructures and their retained nodes act in an identical manner to regular elements and their nodes.

Defining Transformations at Retained Nodes

If a nodal transformation (*Transformed Coordinate Systems*) is used during substructure generation at a retained node, the transformations are built into the substructure. This creates an inconsistency when the substructure node is attached to a standard Abaqus element since Abaqus/Standard uses the retained degrees of freedom directly without checking their directions. Therefore, it is suggested that this situation be avoided.

If a nodal transformation must be used, the resulting inconsistency can be resolved by retaining all degrees of freedom at the node and applying a linear constraint equation (*Linear Constraint Equations*) as follows. At any point where such a transformed substructure node is attached to a global model, define two coincident nodes on the usage level, *P* and *Q*, for example. Use node *P* for the substructure at the usage level (defined with an element definition); the local directions of the degrees of freedom are already built in at this node. Use node *Q* for all standard Abaqus elements attached to this point. Use a local transformation at node *Q* to transform the degrees of freedom to the same local directions that are built-in for node *P*. Now use a linear constraint equation to equate the individual degrees of freedom at nodes *P* and *Q*.

Performing Parametric Studies on the Substructure Stiffness Matrix

You can change the substructure properties to perform parametric studies by controlling the amount of unsymmetry in a substructure stiffness matrix. This feature is allowed only in complex frequency analyses using the unsymmetric solver. You control the unsymmetry by specifying a factor for the unsymmetric part of the substructure stiffness. This feature works only when both a symmetric and an unsymmetric instance of the substructure stiffness matrix are available (see *Generating a Reduced Stiffness Matrix for a Substructure*).

You specify an element set on which you want to perform parametric studies. Abaqus ignores any elements that are not substructures. You specify the factor of unsymmetry, α, in the equation for the stiffness matrix:

$$K = K_s + \alpha(K_u - K_s),$$

where K is the stiffness matrix computed by controlling the stiffness unsymmetry factor, and K_s and K_u are the symmetric and unsymmetric instances, respectively, of the available substructure stiffness matrix.

The default value of the unsymmetry factor is 1.0. You can perform parametric studies on different sets of elements, and the effect of the changes in the stiffness matrix is local to the current step.

Input File Usage: Use the following option to change the substructure properties:

**SUBSTRUCTURE CHANGE*, ELSET=*name*, STIFFNESS UNSYMMETRY FACTOR=α

Abaqus/CAE Usage: Changing the substructure properties is not supported in Abaqus/CAE.

Applying Loads to a Substructure

Loads that are to be applied to a substructure within an analysis (at the usage level) must be specified during the substructure generation step by defining a substructure load case or by requesting that the substructure's gravity load vectors be calculated (see *Defining Substructure Load Cases for Subsequent Loading in an Analysis*). A load case can be made up of any combination of loadings, and multiple load cases can be defined for any given substructure.

When you activate load cases created for a substructure, you specify the element number or element set name of the substructures, the associated substructure load case names, and the scaling multipliers for the specified substructure load case loads. To reproduce the loading conditions defined during substructure generation exactly, use a magnitude of 1.0.

Boundary conditions specified during a substructure's generation are always present. They are effectively built into the substructure and cannot be removed. Boundary conditions cannot be specified within the substructure load cases. See *Generating Substructures* for further information about defining boundary conditions in substructures.

Input File Usage: Use the following option to activate a substructure load case:

**SLOAD*

Abaqus/CAE Usage: Use the following option to activate a substructure load case:

Load module: load editor: **Category**: **Mechanical**: **Types for Selected Step**: **Substructure load**

Modifying or Removing Load Cases

By default, substructure loads are applied as modifications of existing loads or in addition to any loads previously defined. You can remove all previously defined loads and, optionally, specify new loads when you activate a load case. Boundary conditions cannot be removed.

Input File Usage: Use the following option to modify load cases:

**SLOAD*, OP=MOD

Use the following option to remove load cases:

**SLOAD*, OP=NEW

Abaqus/CAE Usage: Use the following option to modify load cases:

Load module: **Load Case Manager**: click **Edit**

Use the following option to remove load cases:

Load module: **Load Case Manager**: click **Delete**

Specifying Time-Dependent Load Cases

The magnitude of substructure loads can be varied with time by referring to an amplitude definition (*Amplitude Curves*).

Input File Usage: Use the following options to define time-dependent load cases:

**AMPLITUDE*, NAME=*amplitude*
**SLOAD*, AMPLITUDE=*amplitude*

Abaqus/CAE Usage: Use the following options to define time-dependent load cases:

Load module: amplitude editor: **Create Amplitude**: **Amplitude**: *amplitude*

Load module: load editor: **Category**: **Mechanical**: **Types for Selected Step**: **Substructure load**: **Amplitude**: *amplitude*

Load Cases in Geometrically Nonlinear Analyses

All substructure loads and boundary conditions are applied in a local system associated with the substructure. Since this local system rotates with the substructure when large motions are present, these loads and boundary conditions will rotate as well. As a consequence, you should be careful when using substructure loads in geometrically nonlinear analyses to ensure that the loading is in the appropriate direction at the usage level. This situation is similar to rotating the substructure via a substructure property definition.

Gravity Loading

A distributed load definition can be used to apply gravity loading to a substructure with a user-defined magnitude, scaled by an amplitude definition, and acting in a specified direction. To enable gravity loading for a substructure, you must request the calculation of the substructure's gravity load vectors during the substructure generation step (see *Gravity Loading*). In this case gravity loading should not be defined as part of a substructure load case.

Input File Usage: Use the following option to define gravity loading:

**DLOAD*, AMPLITUDE=*amplitude*
element set or element number, GRAV, *magnitude, direction*

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Gravity** for the **Types for Selected Step**

Obtaining Displacement Output at the Nodes of the Display Elements Representing Substructures

The display elements are created automatically in the substructure usage analysis for substructures for which display representation has been defined at generation and applied by the substructure property at usage. If you request output of the nodal displacements for all the nodes in the model, it is also obtained at the nodes of display elements. If you do not request output of the nodal displacements for all the nodes in the model, it is recommended to request this output specifically at the nodes of the substructure display elements.

Input File Usage: Use the following options to obtain displacement output at the nodes of the display elements representing substructures:

**OUTPUT*, FIELD
**NODE OUTPUT*, DISPLAY

Abaqus/CAE Usage: Obtaining output at the nodes of the display elements is not supported in Abaqus/CAE.

Obtaining Solutions at the Substructure Monitor Nodes

If you specify a node set (during the substructure generation) for which to monitor the substructure solution, Abaqus/Standard creates the monitor nodes automatically in the substructure usage analysis and applies the substructure property at usage. The created monitor nodes are not connected to the nodes or elements in the usage analysis mesh, and the monitor nodes do not affect the simulation results.

The monitor nodes can have translational displacement degrees of freedom, rotational degrees of freedom, the acoustic pressure degree of freedom, and other degrees that were active in the substructure generation analysis. You cannot apply loads and boundary conditions at the monitor nodes in the usage analysis. You must request output specifically at the substructure monitor nodes for particular analysis steps. Abaqus/Standard creates and applies history output requests for predefined output variables at the monitor nodes automatically (including the translations, rotations, and acoustic pressure for the active degrees of freedom of the monitor nodes). In dynamic analyses (except for implicit integration transient dynamic analyses), Abaqus/Standard also applies velocity and acceleration history output requests at the substructure monitor nodes.

Input File Usage: Use the following options to obtain the solution at the monitor nodes of substructures:

**OUTPUT*, HISTORY
**NODE OUTPUT*, MONITOR

Abaqus/CAE Usage: Obtaining output at the monitor nodes of substructures is not supported in Abaqus/CAE.

Obtaining Output of the Solution for All of the Substructure Degrees of Freedom

The retained nodal degrees of freedom and the generalized degrees of freedom associated with the substructure dynamic modes form the full set of the substructure's degrees of freedom. You can output the solution at all of the substructure degrees of freedom. This feature is available only for Abaqus/Standard transient dynamic analysis; it is not supported for static and linear dynamic analyses. For more information, see *Defining the Retained Nodal Degrees of Freedom* and *Defining the Generalized Degrees of Freedom*.

Input File Usage: Use the following option to output the solution for all of the substructure degrees of freedom:

**SUBSTRUCTURE OUTPUT*

Abaqus/CAE Usage: Obtaining output of the solution for all of the substructure degrees of freedom is not supported in Abaqus/CAE.

Obtaining Output for Selected Substructures

By default, the output is performed for all substructures in the model. You can output the solution for selected substructures by specifying the element set that contains all the substructure-type elements where you want to output the solution.

Input File Usage: Use the following option to obtain output for a selected subset of substructures in the model:

**SUBSTRUCTURE OUTPUT*, ELSET=*element set name*

Abaqus/CAE Usage: Obtaining output of the solution for all of the substructure degrees of freedom is not supported in Abaqus/CAE.

Obtaining Output in Output4 Format

By default, the substructure solution is stored on SIM, which is a high-performance database available in Abaqus. The substructure output data are stored in files named *jobname*`_STEP`*n*`_`*m*`.sim`, where *jobname* is the name of the input file or analysis job, *n* is the number of the Abaqus step that generates the substructure output, and *m* is the substructure element label defined in the input file. The substructure output data written to SIM can later be converted to one of the conventional text or binary formats as a postprocessing operation.

You can also output the substructure solution in Output4 text format, which can be used, for example, by the MSC Nastran finite element solver from MSC.Software Corporation or by the AVL EXCITE™ flexible body dynamics solver from AVL LIST GmbH. The substructure output data in the OP4 text format are stored in files named *jobname*`_STEP`*n*`_`*m*`.op4`, where *jobname* is the name of the input file or analysis job, *n* is the number of the Abaqus step that generates the output, and *m* is the substructure element label defined in the input file.

Input File Usage: Use the following option to output the solution in Output4 text format:

**SUBSTRUCTURE OUTPUT*, FORMAT=OP4

Abaqus/CAE Usage: Obtaining output of the solution for all the substructure degrees of freedom is not supported in Abaqus/CAE.

Obtaining Output of Results within a Substructure

You can obtain output within substructures used in static, dynamic, eigenfrequency extraction, and steady-state and transient modal dynamic analyses. The recovery of output is not possible for substructures used in response spectrum and random response analyses. Output within a substructure does not include the displacements, stresses, etc. resulting from the preload deformation of a substructure. Obtaining output for connector elements in a substructure is not supported in Abaqus/CAE.

Output within substructures is available in the data (`.dat`) file, in the results (`.fil`) file, and in output database (`.odb`) files. Separate output database files are created for each substructure using the naming convention *inputfile-name_substructure-number*`.odb`. If a substructure contains a nested substructure, a file called *inputfile-name_substructure-number_nested-substructure-number*`.odb` is created containing the output for the nested substructure. The **abaqus substructurecombine** execution procedure can combine model and results data from two substructure output databases into a single output database. For more information, see *Combining Output from Substructures*.

Recovery of the solution within substructures requires that the information for recovering the data within a substructure be available from the `.sim`, `.prt`, `.stt`, and `.mdl` files. Availability of the model data (`.odb` or `.sim`) file is not required but is strongly recommended: if this file is not provided, the element results for some element types cannot be displayed in the Visualization module of Abaqus/CAE.

Output is organized substructure by substructure: you direct Abaqus/Standard to go inside a particular substructure and then request output for that substructure. Results can be recovered within nested multilevel substructures only if the substructure databases for all substructures in the chain are available.

Substructure output requests are most easily pictured by thinking of substructures as “levels” of detailed modeling. At the global (top) level we have the analysis model (for example, an airplane). Dropping down from this level to the first substructure level, we have the main components of the model defined as substructures (wings, stabilizer, fuselage, etc.). Dropping down to the second substructure level, we have other substructures (flaps, tanks, floors, etc.), which, in turn, may contain third level substructures (spars, stringers, etc.), and so on. To obtain output, you move down and back up through these various levels using substructure paths, similar to the way you navigate a tree structure for file directories. Each substructure path definition consists of entering into a substructure at the next level down or leaving the current substructure and moving up one level in the tree.

At the start of the output requests, Abaqus/Standard is at the global model level. You must always enter and leave a substructure consistently, so that after a set of substructure output requests Abaqus/Standard is left at the global model level. You must return to the global level (outside all substructures) before the end of the step definition.

If you enter and leave in the same substructure path definition, the effect is to leave the substructure and enter another substructure at the same level.

Entering a Substructure for Output

To enter a particular substructure for output, you identify the substructure by the element number *n* chosen for it in the model. All subsequent output requests are for output within that substructure and must be given in terms of its internal node and element numbers (the node and element numbers used when the substructure was created).

Input File Usage: **SUBSTRUCTURE PATH*, ENTER ELEMENT=*n*

Abaqus/CAE Usage: Step module: field output request editor: **Domain**: **Substructure**: click the **Edit** button, and select substructure sets

Leaving a Substructure after Obtaining Output

After you have obtained output for a substructure, you must return to the level of the model of which the substructure forms a part, thus indicating the end of the output requests for variables within that substructure.

Input File Usage: **SUBSTRUCTURE PATH*, LEAVE

Abaqus/CAE Usage: Step module: field output request editor: **Domain**: **Substructure**: click the **Edit** button, and select substructure sets

Obtaining Output If Substructures Are Nested

You must enter several substructures if substructures are used at multiple levels and output is required several levels down. Nesting of substructures is not supported in Abaqus/CAE.

Example: Obtaining Output within Nested Substructures

For example, suppose that a model includes several substructures at two levels. Printed output of stress components is required in some elements within two substructures at the second level, as well as printed output of the displacements at some of the nodes of one of the first-level substructures. (Recall that "first-level" refers to substructures used directly in the analysis model; "second-level" substructures are used as components of first-level substructures.)

The data might be as follows:

```
*SUBSTRUCTURE PATH, ENTER ELEMENT=N
** This option takes us into element number N, which must be a substructure.
*SUBSTRUCTURE PATH, ENTER ELEMENT=M
** We now drop down into element number M of this substructure.
** M is the element number used for this substructure when N was created.
** M must refer to a substructure.
*EL PRINT, ELSET=A1
S
** This option requests stress output in element set A1 of this substructure.
** This element set must have been defined during the creation of substructure M.
*SUBSTRUCTURE PATH, LEAVE
** This option takes us back up into first-level substructure N.
*SUBSTRUCTURE PATH, ENTER ELEMENT=P
** This option takes us down into element P, which must again be a substructure in element N.
*EL PRINT, ELSET=A1
S
** This option requests the printing of stress output in element set A1. It is possible that
** this is the same set of elements in the same substructure as was used in the request above
** because substructures M and P may both be copies of the same substructure.
** However, the stresses will presumably be different because they represent the same
** component in different locations in the model.

*SUBSTRUCTURE PATH, LEAVE
** Back to N.
*SUBSTRUCTURE PATH, LEAVE
** We are now back at the global level.
*SUBSTRUCTURE PATH, ENTER ELEMENT=R
** Enter element R at the global level: this element is the substructure in which we want
** to print the displacements.
*NODE PRINT, NSET=FLANGE
```

```
U
** This option prints the displacements at all nodes in node set
** FLANGE of the substructure.
** Again, FLANGE must have been defined when the substructure was
** created.
*SUBSTRUCTURE PATH, LEAVE
** Back to the global level.
```

Interpreting Nodal Variable Output

The nodal displacements within the substructure do not include the displacements resulting from the preload deformation if it exists.

If a substructure is rotated and/or reflected, nodal variables are output relative to the global coordinate system of the analysis. In a geometrically nonlinear analysis, the nodal displacements will include the large motions associated with the translation and rotation of the substructure in addition to the small-strain displacements. If a nodal transformation (*Transformed Coordinate Systems*) has been used, nodal output will be in either the local or the global directions, depending on the nodal output request (see *Output to the Data and Results Files*). If a nodal transformation has been used during substructure generation, the transformed directions are rotated with the substructure.

Interpreting Element Variable Output

Element output variables within a substructure do not include the values of the variable resulting from the preload deformation if it exists.

Element variables in continuum elements are output relative to the global coordinate system of the analysis model or in the local (material) coordinate system if one has been used (*Orientations*). Element output for structural elements is always given with respect to the element coordinate system used during substructure generation. Integration point coordinates and local material directions (see *Output to the Data and Results Files*) are given with respect to the global coordinate system.

Element quantities associated with nonlinear preload response (plastic strains, creep strains, etc.) can be output during a substructure recovery. Since the response in a substructure during its usage is entirely linear, these quantities, which are part of the base state, do not change from the values computed during the preload.

If a substructure was reflected, the element connectivities of continuum elements written to the substructure instance output database are adjusted so as not to violate the Abaqus convention for counterclockwise element numbering.

You cannot directly obtain the element output for the element centroidal values or the element output at the element nodes when you recover results within substructures. This output data can be calculated from the substructure-related data in the output database file using commands in the Abaqus Scripting Interface.

Interpreting Results Written to the Results File

Results within substructures can be written to the results file. Substructure path records are inserted in the results file to indicate the switch into a substructure: all records following such a record belong to the substructure defined on that record until the next substructure path record appears in the file.

Requests for output to the results file will cause Abaqus/Standard to write the definitions of elements and nodes at the global level and within all substructures in the model to the file. As with the results records themselves, these records for nodes and elements within substructures will be preceded and followed by substructure path records to indicate that they belong to that substructure.

Node and element numbers within each substructure are local to that substructure, so that the same node and element numbers may appear in several substructures and in the global level model. In such a case the substructure path records must be used to identify the location of a particular node or element within the model. If you can

ensure that node and element numbers are unique throughout the entire model, including all substructures, the substructure path records in the results file can be ignored.

Visualizing Substructure Results

While Abaqus/CAE does not support substructures directly, you can view substructure results by combining all of the substructure instance output database (`.odb`) files into a single file. See *Combining Output from Substructures* for details.

You can also load and view each individual substructure instance output database (`.odb`) file separately in Abaqus/CAE.

Substructure Compatibility

Only the substructure SIM database (`.sim`) and the model data (`.odb` or `.sim`) files are backward compatible. If these files were generated from a previous general release or from a previous maintenance delivery of the same general release, they can be upgraded to the current release (see *SIM Database Utilities* and *Output Database Upgrade Utility*). Other substructure files (including the `.stt`, `.prt`, and `.mdl` files) are not compatible between maintenance deliveries of the same general release. These files are required only if recovering results within substructures is performed in the substructure usage analysis. When recovering results, you must regenerate your substructures in the current release.

Input File Template

The following template can be used to generate a substructure:

```
*HEADING
…
*NODE,NSET=N1
Data lines to define the nodes.
…
*NSET,NSET=N3
Data lines to define the node set members.
…
*ELEMENT, TYPE=CPE8, ELSET=E1
Data lines to define the elements that make up the substructure.
…
*ELSET,ELSET=E3
Data lines to define the element set members.
…
*SOLID SECTION, ELSET=E1, MATERIAL=M1
*MATERIAL, NAME=M1
*ELASTIC
30.E6, 0.3
*DENSITY
0.0007324
*STEP
*FREQUENCY
Data line to specify the number of modes ( ≥ m). The *FREQUENCY option
is required if modes are requested using the *SELECT EIGENMODES option.
*END STEP
*STEP
*STATIC
…
Options to define a linear or nonlinear static preload.
…
*END STEP
*STEP
```

```
*SUBSTRUCTURE GENERATE, NAME=MY_SUBSTRUCTURE, OVERWRITE, MASS MATRIX=YES,
VISCOUS DAMPING MATRIX=YES, STRUCTURAL DAMPING MATRIX=YES,
RECOVERY MATRIX=YES, NSET=N3, ELSET=E3
*RETAINED NODAL DOFS
Data lines to define the retained degrees of freedom.
*SELECT EIGENMODES, GENERATE
1, m, 1
*SUBSTRUCTURE LOAD CASE, NAME=LOADS
*CLOAD
Data lines to define concentrated loading.
*DLOAD
Data lines to define distributed loading.
*END STEP
```

The following template can be used to define substructure instances:

```
*HEADING
…
*ELEMENT, TYPE=SUBSTR, FILE=MY_SUBSTRUCTURE, ELSET=E2
Data line to define the element.
*SUBSTRUCTURE PROPERTY, ELSET=E2
*BOUNDARY
…
*RESTART, WRITE
*STEP
*STATIC
…
*BOUNDARY
…
*SLOAD
E2, LOADS, scale factor
*SUBSTRUCTURE PATH, ENTER ELEMENT=n
*EL PRINT
S,
*NODE PRINT
U,
*SUBSTRUCTURE PATH, LEAVE
*END STEP
*STEP
*DYNAMIC
…
*BOUNDARY
…
*SUBSTRUCTURE PATH, ENTER ELEMENT=n
*EL PRINT
S,
*NODE PRINT
U, V
*SUBSTRUCTURE PATH, LEAVE
*END STEP
```

Generating Substructures

Products: Abaqus/Standard Abaqus/CAE

References:

- *Using Substructures*
- **RETAINED NODAL DOFS*
- **SELECT EIGENMODES*
- **SUBSTRUCTURE GENERATE*
- **SUBSTRUCTURE LOAD CASE*
- **SUBSTRUCTURE MATRIX OUTPUT*
- **MATRIX CHECK*
- **FLEXIBLE BODY*

Overview

This section describes how to define individual conventional substructures.

You can also define frequency-based substructures (see *Generating Frequency-Based Substructures*) for use in direct steady-state dynamic analyses. Frequency-based substructures differ from conventional substructures in that they use stiffness, inertial, damping, and frequency information to form the complex, frequency-domain analysis operator while conventional substructures compute real, condensed substructure operators.

For information on how substructures are used in a model, see *Using Substructures*.

Substructures are defined using the substructure generation procedure. The substructure creation and usage cannot be included in the same analysis. Multiple substructures can be generated in an analysis. Any substructure can consist of one or more other substructures; if this is the case, the nested-level substructures must be defined first. The substructure database is not organized in terms of part instances; therefore, substructures cannot be generated from models that have an assembly defined. None of the substructure options are supported in models that have an assembly defined.

To define a typical substructure generation step, do the following:

- Invoke the substructure generation procedure.
- Define the nodes and degrees of freedom that are to be retained as external degrees of freedom when the substructure is used.
- Optionally, retain extra dynamic modes to improve the dynamic behavior of the substructure during usage.
- Optionally, specify substructure load cases.
- Optionally, write the recovery matrix, substructure's stiffness matrix, mass matrix, and load case vectors to a file.

Generating a Substructure

The generated substructure is associated with a name. You can specify the substructure name, which is the preferred method for naming a substructure. By default, a substructure is named *jobname*_Z*n*, where *jobname* is the name of the substructure generation job, and *n* is the current step number of the job.

The substructure name must be unique in the substructure generation analysis. If several substructures are generated in the same substructure generation job, they must have different substructure names.

Input File Usage: Use the following option to specify the substructure name:

**SUBSTRUCTURE GENERATE*, NAME=*name*

Abaqus/CAE Usage: Specifying the substructure name is not supported in Abaqus/CAE.

Alternative Method for Naming a Substructure

Abaqus provides an alternative method to name a substructure. Instead of using the default name or specifying the substructure name, you can specify a prefix and an identifier. The identifier must begin with the capital letter Z followed by a positive integer that cannot exceed 9999. You can specify *libname* as the prefix and Z*n* as the identifier to construct the substructure name *libname*_Z*n*.

Input File Usage: The alternative way to define a substructure database name:

**SUBSTRUCTURE GENERATE*, TYPE=Z*n*, LIBRARY=*libname*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: *n*

Substructure Database

A substructure database is the set of files that describes the mechanical and geometric properties of a substructure. Abaqus writes all generated substructure data to the substructure database during the substructure generation analysis. The substructure database can include files with the following extensions: `.sim`, `.prt`, `.mdl`, `.stt`, and `.odb`.

The substructure database files are named using the substructure name. Therefore, if several substructures are generated in the same substructure generation job, they must have different substructure names. If a substructure with the same name already exists in the user directory, the analysis ends with an error message unless you specify to overwrite the existing substructure database files (see *Overwriting the Substructure Database Files*).

The substructure files are named as follows:

- *name*`.sim`
- *name*`.prt`
- *name*`.mdl`
- *name*`.stt`
- *name*`_MODEL.odb`
- *name*`_MODEL.sim`

The *name*`.sim` file is generated for every substructure. This file contains the condensed substructure operators, recovery matrices, and other generated substructure entities.

Files with the extensions `.prt`, `.mdl`, and `.stt` contain the internal Abaqus database for the finite element model from which a substructure is generated. These files are generated only if the solution within the substructure can be fully or partially recovered. Abaqus uses these files to recover element results within the substructure in the substructure usage analyses.

Model Data Files

The *name*`_MODEL.odb` and *name*`_MODEL.sim` files are the model data files. You can use the **abaqus** execution procedure to specify the output format of the results (see *Abaqus/Standard and Abaqus/Explicit Execution*). By default, the *name*`_MODEL.odb` file is generated if the following conditions are satisfied:

- You run the **abaqus** execution procedure using the **resultsformat**=`odb` (default) command line option.

- The substructure is generated from a three-dimensional finite element model.
- The solution within the substructure can be fully or partially recovered.

The *name*_MODEL.odb file contains the finite element model data required for visualization of the results recovered within the substructure. You can suppress generation of this file. If a substructure does not include this file, you can still visualize the recovered substructure results in the Visualization module of Abaqus/CAE; however, the element results (for example, stresses) cannot be displayed for some groups of elements (for example, beam and shell elements). The *name*_MODEL.odb file cannot be created if a substructure is generated from a two-dimensional finite element model.

The *name*_MODEL.sim file is generated if you run the **abaqus** execution procedure using the **resultsformat**=sim command line option. There are no limitations related to the space dimensions for this file generation. You can visualize the substructure results using the Physics Results Explorer app on the **3D**EXPERIENCE platform.

If you run the **abaqus** execution procedure using the **resultsformat**=both command line option, both the *name*_MODEL.odb and *name*_MODEL.sim files are created for a substructure generated with recovery enabled.

Input File Usage: Use the following option to specify the output format of the model data file in the input file rather than in the execution procedure:

**SUBSTRUCTURE GENERATE*, MODEL DATA=ODB

Use the following option to suppress generation of the model data file:

**SUBSTRUCTURE GENERATE*, MODEL DATA=NONE

Abaqus/CAE Usage: Controlling the generation of the model data file is not supported in Abaqus/CAE.

Overwriting the Substructure Database Files

If a substructure generation analysis is rerun using the same substructure name without deleting the substructure database files, you must specify that the existing substructure database files can be overwritten.

Input File Usage: **SUBSTRUCTURE GENERATE*, OVERWRITE

Abaqus/CAE Usage: Specifying that the existing substructure database files can be overwritten is not supported in Abaqus/CAE.

Renaming the Substructure Database Files

You can rename the files in a substructure database. You must rename all of the files consistently with the new name. For example, replace the substructure *name* with a *newname* to rename the files as follows:

- *newname*.sim
- *newname*.prt
- *newname*.mdl
- *newname*.stt
- *newname*_MODEL.odb
- *newname*_MODEL.sim

Recovery within a Substructure

By default, the solution at any degree of freedom in the substructure can be recovered. Abaqus must have access to the substructure's `.mdl`, `.prt`, and `.stt` files to perform a full recovery. These files all reside in the substructure database.

You can specify that a recovery of element or nodal information will not be required within this substructure. This reduces the size of the substructure database significantly for a large substructure because the information that is required to recover eliminated variables is not stored. However, this information cannot be recreated later except by regenerating the entire substructure with recovery enabled.

Input File Usage: Use the following option to enable recovery for a substructure:

**SUBSTRUCTURE GENERATE*, RECOVERY MATRIX=YES (default)

Use the following option to disable recovery for a substructure:

**SUBSTRUCTURE GENERATE*, RECOVERY MATRIX=NO

Abaqus/CAE Usage: Use the following option to enable recovery for a substructure:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Whole model**

Use the following option to disable recovery for a substructure:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle off **Evaluate recovery matrix for**

Recovery Method for Nodal Displacement

If results recovery is desired only at a subset of the internal degrees of freedom, disk usage can be reduced substantially by using the selective recovery method. To enable selective recovery, the region where recovery is desired can be specified directly.

Input File Usage: Use the following option to define the node set for selective recovery:

**SUBSTRUCTURE GENERATE*, RECOVERY MATRIX=YES, NSET=*Node set name*

Use the following option to define the element set for selective recovery:

**SUBSTRUCTURE GENERATE*, RECOVERY MATRIX=YES, ELSET=*Element set name*

Abaqus/CAE Usage: Use the following option to define the node set for selective recovery:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Region**: *Node set name*

Use the following option to define the element set for selective recovery:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic**
tabbed page: toggle on **Evaluate recovery matrix for**: select **Region**: *Element set name*

Recovery Method for Element Results

You can use the **substructurerecover** utility (see *Recovering Results from a Substructure*) to recover the solution in the substructure. This utility performs results recovery for static or dynamic time-domain or frequency-domain solutions obtained from Abaqus/Standard and from other workflows using Abaqus substructures.

The **substructurerecover** utility recovers results within a substructure using the recovery matrices. To use this recovery method, you must have a full or selective recovery matrix sufficient for recovering the nodal output variables. To enable element results recovery with this method, the element recovery matrices should be generated for a substructure.

To use the **substructurerecover** utility, the substructure model data file (see *Model Data Files*) must be available, and the solution for all the substructure interface degrees of freedom (see *Obtaining Output of the Solution for All of the Substructure Degrees of Freedom*) must be provided.

You can customize element results recovery within a substructure by specifying multiple element recovery matrix generation options in a single substructure generation analysis step.

Input File Usage: Use the following option to enable element recovery matrices for a substructure:

**ELEMENT RECOVERY MATRIX*

Use the following option to specify one of the supported element output positions for recovering substructure results:

**ELEMENT RECOVERY MATRIX*, POSITION=*Position type*

Use the following option to request element recovery matrix generation for a part of the model:

**ELEMENT RECOVERY MATRIX*, ELSET=*Element set name*

Use the following option to request element recovery matrix generation only for the exterior three-dimensional elements:

**ELEMENT RECOVERY MATRIX*, EXTERIOR

Abaqus/CAE Usage: Controlling the generation of the element recovery matrices is not supported in Abaqus/CAE.

Evaluating Frequency-Dependent Material Properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in substructure generation. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness at zero frequency and does not consider the stiffness contributions from frequency-domain viscoelasticity. If you do specify a frequency, only the real part of the stiffness contributions from frequency-domain viscoelasticity is considered.

Input File Usage: **SUBSTRUCTURE GENERATE*, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: Step editor: **Substructure generate**: **Options** tabbed page: toggle on **Evaluate frequency-dependent properties at frequency**: *frequency*

Defining the Retained Nodal Degrees of Freedom

The degrees of freedom at a node can be divided into retained degrees of freedom (for use at the usage level of the substructure) and eliminated degrees of freedom (internal to the substructure). Abaqus/Standard allows any of the degrees of freedom at any of the nodes of a substructure to be retained with one exception: if an acoustic-structural substructure is generated, based on coupled or uncoupled modes, only structural degrees of freedom can be retained. You must make sure that the choice of retained degrees of freedom is reasonable so that the substructure can be connected correctly to the rest of the model.

Any degrees of freedom where kinematic constraints might have to be respecified during usage of the substructure should be kept as retained degrees of freedom.

If any degrees of freedom of nodes used to define distributing coupling elements are retained, the degrees of freedom of an internal node associated with the Lagrange multipliers are added automatically to the list of the retained degrees of freedom of the substructure.

To define the retained degrees of freedom, specify the node number or node set label and, optionally, the first and the last degree of freedom to be retained.

By default, the nodes associated with the retained degrees of freedom will be sorted into ascending numerical order.

Input File Usage: **RETAINED NODAL DOFS*

Abaqus/CAE Usage: Load module: boundary condition editor: **Category**: **Mechanical**: **Types for Selected Step**: **Retained nodal dofs**

Preventing the Degrees of Freedom from Being Sorted

You can prevent the degrees of freedom from being sorted. The ordering of the nodes when using a substructure is then the same as the ordering used when specifying the retained nodes.

Input File Usage: **RETAINED NODAL DOFS*, SORTED=NO

Abaqus/CAE Usage: You cannot prevent retained nodes from being sorted in Abaqus/CAE.

Retaining Degrees of Freedom When the Substructure Is Intended for Geometrically Nonlinear Analysis at the Usage Level

When the substructure is intended for use in geometrically nonlinear analyses, it is recommended to retain all translational and/or all rotational degrees of freedom from a particular node. Even in the case when only a single translational/rotational degree of freedom of a particular node is deemed as needed at the usage level, you should retain all translational/rotational degrees of freedom associated with that node. Otherwise, as the substructure rotates during a geometrically nonlinear analysis, local numerical instabilities (negative eigenvalues) might occur since the rotated substructure might have no stiffness in particular degrees of freedom.

You must choose an appropriate number of nodes that will allow for the computation of an equivalent rigid body motion of the substructure. In two-dimensional or axisymmetric analyses, retaining two nodes with all translational degrees of freedom or one node with all translational and rotational degrees of freedom is sufficient to compute an equivalent rigid body motion of the substructure at the usage level. In three-dimensional analysis, three non-collinear nodes with all translational degrees of freedom retained or one node with all translations and

rotations are needed. If the retained nodes are collinear or fewer than three nodes are retained, you must retain at least one node with all rotational degrees of freedom. When Abaqus/Standard cannot compute an equivalent rigid body motion for the substructure during the analysis at the usage level because the number of retained degrees of freedom is not appropriate, a warning message is issued and any geometrically nonlinear effects associated with the substructure are ignored.

Defining Kinematic Constraints

Kinematic constraints are defined as described in *About Kinematic Constraints*. The following rules apply:

- All kinematic boundary conditions associated with degrees of freedom that are not retained must be specified when the substructure is generated. The conditions are built into the substructure and remain imposed any time that it is used. Once the substructure is generated, kinematic constraints on internal variables cannot be respecified; they can be modified or removed only by recreating the substructure. The magnitude of a prescribed boundary condition applied to an internal degree of freedom can be associated with a substructure load case and can be changed at the usage level. The restraint itself is built into the substructure and cannot be removed by omitting a reference to the load case.
- During substructure generation, multi-point constraints in which some of the substructure's retained degrees of freedom are eliminated in favor of internal degrees of freedom must be avoided. If it is desirable to retain certain degrees of freedom that are eliminated by the multi-point constraints, you must reassign all the variables appearing in the multi-point constraints as retained degrees of freedom and impose the constraints at the usage level.

Defining the Generalized Degrees of Freedom

An effective technique for modeling the dynamic behavior of a substructure is to augment the response within the substructure by including some generalized degrees of freedom associated with the dynamic modes. You can select the modes to retain, which must be calculated in a previous frequency extraction step (*Natural Frequency Extraction*). For some cases of the substructure generation, the dynamic modes have to be fully recovered; if they were computed with the AMS eigensolver and only partially recovered, an error message is issued in such cases. For example, if a substructure includes the substructure load cases or structural-acoustic coupling (or it will be used for flexible body generation) the eigenmodes have to be fully recovered. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. If all retained degrees of freedom of the substructure are constrained in the frequency extraction step, this technique is commonly referred to as the Craig-Bampton method. If all retained degrees of freedom of the substructure are not constrained in the frequency extraction step, this technique is commonly referred to as the Craig-Chang method. The substructure dynamic modes in the Craig-Bampton method are commonly referred to as the fixed-interface modes, and the substructure dynamic modes in the Craig-Chang method are commonly referred to as the free-interface modes. If some retained degrees of freedom of the substructure are constrained and other retained degrees of freedom are not constrained in the frequency extraction step, the dynamic modes are called mixed-interface modes. If the free-interface or mixed-interface dynamic modes are selected, the substructure generation time can increase substantially compared to the case when the same number of fixed-interface dynamic modes is used. Abaqus issues a warning message in this case. However, better solution accuracy can sometimes be achieved with a significantly smaller number of free- or mixed-interface dynamic modes than by using fixed-interface modes.

A sufficient number of the dynamic modes should be selected to provide adequate dynamic representation of the substructure. You should examine loading frequencies and frequency content of the structure to determine this range. Specify a shift point and/or a cutoff frequency in the eigenfrequency extraction step definition to obtain modes in the desired frequency range only. Inclusion of generalized degrees of freedom adds the cost of the frequency extraction to the substructure generation step but greatly improves the accuracy of the solution if the substructure is used in a subsequent dynamic (*Implicit Dynamic Analysis Using Direct Integration*), steady-state dynamic (*Direct-Solution Steady-State Dynamic Analysis*), or frequency extraction (*Natural Frequency Extraction*) analysis.

In the case of the displacement normalization of the eigenvectors in a frequency extraction analysis, a substructure must have at least one physical degree of freedom active on the usage level; otherwise, the modes cannot be normalized properly. See *Substructuring and substructure analysis* for additional details.

The retained eigenmodes must be selected when an acoustic-structural substructure is generated.

The effect of acoustic-structural coupling can be included in the retained eigenmodes during the natural frequency extraction procedure. To calculate the coupled structural-acoustic eigenmodes, use a frequency extraction analysis with the default Lanczos eigensolver and include the effect of acoustic-structural coupling during the natural frequency extraction procedure (*Natural Frequency Extraction*).

Abaqus can also use uncoupled eigenmodes, generated from either SIM-based Lanczos or AMS eigensolver, to generate a coupled acoustic-structural substructure. In this case the effect of acoustic-structural coupling is included during the substructure generation. Both structural and acoustic eigenmodes have to be retained for the substructure generation, and the selection of the acoustic zero-frequency modes, if such modes are present, is required to get an accurate substructure.

Selecting the Modes to Be Used in a Substructure Generation Analysis by Their Mode Numbers

You can directly specify the eigenmodes to be used in a substructure generation analysis by their mode numbers.

Input File Usage: **SELECT EIGENMODES*
eigenmode 1, *eigenmode 2*, *etc.*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by mode range, with each row in the data table specifying a single mode number. The starting mode number and ending mode number in each row should be equal, and the increment value should be zero.

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Mode range**: **Start Mode**: *eigenmode 1*: **End Mode**: *eigenmode 1*: **Increment**: 0**Start Mode**: *eigenmode 2*: **End Mode**: *eigenmode 2*: **Increment**: 0*etc.*

Generating a List of the Eigenmodes by Mode Range

Instead of listing all the retained eigenmode numbers, you can generate the list of eigenmodes.

Input File Usage: Use the following option to generate the list of eigenmodes by mode range, with each data line specifying the start mode number, the end mode number, and the increment in mode numbers between these two values:

**SELECT EIGENMODES*, GENERATE
first mode number, *last mode number*, *increment*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by mode range, with each row in the data table specifying the start mode number, the end mode number, and the increment in mode numbers between these two values:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Mode range**: **Start Mode**: *first mode number*: **End Mode**: *last mode number*: **Increment**: *increment*

Generating a List of the Eigenmodes by Frequency Range

You can select all the modes from the specified frequency range including frequency boundaries.

Input File Usage: Use the following option to generate the list of eigenmodes by frequency range, with each data line specifying the lower boundary of the frequency range and the upper boundary of the frequency range:

**SELECT EIGENMODES*, DEFINITION=FREQUENCY RANGE
lower boundary of the frequency range, *upper boundary of the frequency range*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by frequency range, with each row in the data table specifying the lower boundary of the frequency range and the upper boundary of the frequency range:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Frequency range**: **Lower Frequency**: *lower boundary of the frequency range*: **Upper Frequency**: *upper boundary of the frequency range*

Defining the Substructure Display Representation

In a substructure generation analysis, you can specify a set of elements to define the coarse geometric representation of the substructure in the usage analysis. The nodes of these elements must have displacement degrees of freedom. This element set can include any 2-node elements (truss, beam, connector, etc.) and 3- or 4-node elements with two-dimensional geometry (surface, membrane, shell).

To define the substructure coarse geometric shape, you can specifically add elements that do not change the mechanical behavior of the model (for example, connector elements not associated with any connector behavior definition, surface elements, etc.). If these elements introduce new nodes rather than using nodes from existing elements, the new nodes must be connected to existing elements with significant mechanical properties. In this way, the new nodes used for defining the geometric shape also follow the model deformation.

Input File Usage: Use the following option to specify the element set for the substructure display representation:

**SUBSTRUCTURE GENERATE*, DISPLAY ELSET=*element_set_name*

Abaqus/CAE Usage: Specifying the display element set is not supported in Abaqus/CAE.

Monitoring the Substructure Solution

In a substructure generation analysis, you can specify a set of nodes for which to monitor the solution within the substructure in the usage analysis. The selected nodes can have any degrees of freedom (for example, translations, rotations, acoustic pressure, etc.).

Input File Usage: Use the following option to specify the node set for which to monitor the substructure solution:

**SUBSTRUCTURE GENERATE*, MONITOR NSET=*node_set_name*

Abaqus/CAE Usage: Specifying the monitor node set is not supported in Abaqus/CAE.

Substructure Size

Abaqus limits the substructure size to 16,384 degrees of freedom (including retained nodal and generalized degrees of freedom) for substructures used in Abaqus and to 46,340 degrees of freedom for substructures generated in Abaqus and used outside of Abaqus, such as for flexible body dynamics workflows. Abaqus exits with an error message if you request generation of a substructure with more than 46,340 degrees of freedom.

Preloading a Substructure

Substructures can be used in models that exhibit nonlinear response (associated with standard Abaqus elements or with contact definitions), but the response within a substructure assumes linear small deformations. However, a substructure's response might be a linear perturbation about a predeformed (possibly rotating and translating) base state, defined on the basis of nonlinear response within the substructure during its preload history.

When the substructure is intended for use in geometrically nonlinear analyses, the substructure preloading should be limited to loads that generate self-equilibrating stresses only (such as thermal stresses or interference fits). In most cases, preload stresses are not self-equilibrating (such as stresses from specified boundary conditions or applied loads). If non-self-equilibrating prestress exists and the substructure undergoes a rigid body motion at the usage level, additional stress is generated in the substructure. Such usage level stresses are nonphysical and will lead to convergence problems and results that are difficult to interpret. Therefore, you should use extreme care when preloading a substructure intended for use in geometrically nonlinear analyses.

This preloading concept allows such effects as stress stiffening to be included in a substructure. Preloading is a part of the state of the substructure: the preload is self-equilibrating and so does not generate a load vector when the substructure is used. Any loading of the substructure during its use in a model is in addition to the preload.

It is important to distinguish the difference between a preload and a load case. Both are allowed during a substructure generation analysis, but only the preloads are actually applied to the substructure during generation. Load cases, defined during substructure generation, can only be applied at the usage level (see *Applying Loads to a Substructure*). Load cases are discussed in more detail later.

Computation of the Total Response of a Variable

Any recovered response variable within a substructure (such as stress or displacement) is defined to be a perturbation (with some exceptions for geometrically nonlinear analyses) from the preloaded base state. For geometrically nonlinear analyses, the displacement output includes both the equivalent rigid body rotation and translation associated with the substructure and the strain-inducing small-displacement perturbation. If the total response of a variable is desired, it can be computed by adding the perturbation result to the final result computed during the substructure preload.

Computation of the Tangent Stiffness of a Preloaded Substructure

The rules for calculating the stiffness matrix of a preloaded substructure are the same as those for a static linear perturbation step. See *General and Perturbation Procedures* for a detailed description of the rules.

Defining a Preloading History

Specify the loading history that defines the preload state for a substructure.

Input File Usage: Use the following options:

**STEP*
Options to define the preloading history.
**END STEP*
Any number of steps can be defined.
**STEP*

**SUBSTRUCTURE GENERATE*
Options to define the substructure.
**END STEP*

Abaqus/CAE Usage: The **Substructure generation** step must be defined after the preloading steps in an Abaqus/CAE analysis.

Prescribing Boundary Conditions at Retained Degrees of Freedom during Preloading Steps

During substructure preloading, boundary conditions can be prescribed at retained degrees of freedom. When the preloaded substructure is subsequently created in a substructure generation step, you must release all the retained degrees of freedom (see *Removing Boundary Conditions*). An error message will be issued if some of the retained degrees of freedom are not released. The reaction forces at the released degrees of freedom become concentrated loads that are in equilibrium with the stresses within the substructure. These concentrated loads cannot be removed without changing the preload.

The preloaded substructure is, thus, in equilibrium. If the preload in a substructure must effectively apply loading to other parts of the structure, a substructure load case corresponding to the loads applied in the preload history must be created.

The technique is demonstrated in *Analysis of a rotating fan using substructures and cyclic symmetry*.

Generating a Reduced Stiffness Matrix for a Substructure

You can generate a reduced stiffness matrix for a substructure. The default behavior is based on whether Abaqus/Standard uses the symmetric or unsymmetric solver for the current analysis step (see *Matrix Storage and Solution Scheme in Abaqus/Standard*). A symmetric instance of the substructure's reduced stiffness matrix is generated when Abaqus/Standard uses the symmetric solver. An unsymmetric instance of the substructure's reduced stiffness matrix is generated when Abaqus/Standard uses the unsymmetric solver.

You can modify this behavior to generate a symmetric instance, an unsymmetric instance, or both a symmetric and an unsymmetric instance of the reduced stiffness matrix regardless of whether Abaqus/Standard uses the symmetric or unsymmetric solver. For acoustic-structural substructures, you can modify the default behavior only for those substructures generated using coupled modes.

If the global stiffness matrix of the model involves any unsymmetry, the symmetric instance of the reduced stiffness matrix generated in a step using the unsymmetric solver might not be equivalent to the symmetric instance of the reduced stiffness matrix generated in a step using the symmetric solver. The same is true about the unsymmetric instance of the reduced stiffness matrix generated in a step using the symmetric solver and the unsymmetric instance of the reduced stiffness matrix generated in a step using the unsymmetric solver.

For models with some sources of unsymmetry in stiffness (for example, models including sliding contact with friction), generating both the symmetric and unsymmetric instances of the reduced stiffness matrix can be beneficial at the usage level. Benefits at the usage level include the following:

- When a substructure generated with only an unsymmetric reduced stiffness matrix is used in an eigenfrequency analysis, Abaqus/Standard performs an averaging-based symmetrization of the unsymmetric stiffness matrix. The symmetric matrix is used for frequency extraction, and the eigenfrequencies that the procedure yields can be unphysical for some models. However, when a substructure generated with both the symmetric and unsymmetric instances of the stiffness matrix is used, appropriate instances of the stiffness matrix are chosen for the procedures in which you use the substructure. For example, an eigenfrequency procedure uses the symmetric instance of the stiffness matrix, and a static analysis using the unsymmetric solver uses the unsymmetric instance of the stiffness matrix.
- When a substructure generated with both the symmetric and unsymmetric instances of the stiffness matrix is used in a complex frequency extraction procedure, you can perform parametric studies by using a stiffness

matrix obtained from a linear combination of the symmetric and unsymmetric instances of the substructure's stiffness matrix.

Input File Usage: Use the following option to generate a symmetric instance of the reduced stiffness matrix:

**SUBSTRUCTURE GENERATE*, STIFFNESS MATRIX=SYMMETRIC

Use the following option to generate an unsymmetric instance of the reduced stiffness matrix:

**SUBSTRUCTURE GENERATE*, STIFFNESS MATRIX=UNSYMMETRIC

Use the following option to generate both a symmetric and an unsymmetric instance of the reduced stiffness matrix:

**SUBSTRUCTURE GENERATE*, STIFFNESS MATRIX=BOTH

Abaqus/CAE Usage: Generating a reduced stiffness matrix for a substructure is not supported in Abaqus/CAE.

Generating a Reduced Mass Matrix for a Substructure

You can generate a reduced mass matrix for a substructure.

A reduced mass matrix is calculated by projecting the global mass matrix to the subspace of the substructure modes. This technique is known as Guyan reduction if only the static modes associated with the nodal retained degrees of freedom are used. Using only the static modes might not be sufficient to define the dynamic response of the substructure accurately. Additional dynamic modes must be used to improve the response inside the substructure.

Input File Usage: **SUBSTRUCTURE GENERATE*, MASS MATRIX=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced mass matrix**

Generating a Reduced Viscous Damping Matrix for a Substructure

Viscous damping in the Abaqus model can be defined by "Rayleigh-type" damping associated with materials (see *Material Damping*), by dashpots (see *Dashpots*), by connector elements, by user-defined elements, by direct matrix input (see *Using Matrices*), and by some other modeling features. You can generate a reduced structural damping matrix for a substructure that will represent all sources of the viscous damping in the model.

The reduced viscous damping matrix is calculated in a manner similar to that used for the reduced mass matrix.

Input File Usage: **SUBSTRUCTURE GENERATE*, VISCOUS DAMPING MATRIX=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced viscous damping matrix**

Friction Damping Effects

Friction at the contact nodes, at which a velocity differential is imposed, can give rise to the viscous damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with the velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see *Coulomb friction*. You can include these friction-induced contributions to the reduced viscous damping matrix.

Input File Usage: **SUBSTRUCTURE GENERATE*, VISCOUS DAMPING MATRIX=YES, FRICTION DAMPING=YES

Generating a Reduced Structural Damping Matrix for a Substructure

Structural damping in the Abaqus model can include contributions from the material structural damping defined as a scaling factor for the stiffness (the imaginary stiffness), damping contributions from frequency-domain viscoelasticity, structural damping contributions from connectors and spring elements, and from user-defined elements. It can also be defined by direct matrix input (see *Using Matrices*). You can generate a reduced structural damping matrix for a substructure.

The reduced structural damping matrix is calculated in a manner similar to that used for the reduced mass matrix.

A symmetric instance of the reduced structural damping matrix is generated when Abaqus/Standard uses the symmetric solver. An unsymmetric instance of the reduced structural damping matrix is generated when Abaqus/Standard uses the unsymmetric solver. For more information, see *Matrix Storage and Solution Scheme in Abaqus/Standard*).

You can generate a symmetric instance, an unsymmetric instance, or both a symmetric and an unsymmetric instance of the reduced structural damping matrix regardless of whether Abaqus/Standard uses the symmetric or unsymmetric solver. For acoustic-structural substructures, you can generate these instances only for substructures generated using coupled modes.

If the global stiffness matrix of the model involves any unsymmetry, the symmetric instance of the reduced structural damping matrix generated in a step using the unsymmetric solver might not be equivalent to the symmetric instance of the reduced structural damping matrix generated in a step using the symmetric solver. The same is true about the unsymmetric instance of the reduced structural damping matrix generated in a step using the symmetric solver and the unsymmetric instance of the reduced structural damping matrix generated in a step using the unsymmetric solver.

Input File Usage: Use the following option to calculate the substructure's reduced structural damping matrix:

**SUBSTRUCTURE GENERATE*, TYPE=Z*n*, STRUCTURAL DAMPING MATRIX=YES

Use the following option to generate a symmetric instance of the reduced structural damping matrix:

**SUBSTRUCTURE GENERATE*, STRUCTURAL DAMPING MATRIX=SYMMETRIC

Use the following option to generate an unsymmetric instance of the reduced structural damping matrix:

*SUBSTRUCTURE GENERATE, STRUCTURAL DAMPING MATRIX=UNSYMMETRIC

Use the following option to generate both a symmetric and an unsymmetric instance of the reduced structural damping matrix:

*SUBSTRUCTURE GENERATE, STRUCTURAL DAMPING MATRIX=BOTH

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced structural damping matrix**

Generating a symmetric instance, an unsymmetric instance, or both a symmetric and an unsymmetric instance of the reduced structural damping matrix regardless of whether Abaqus/Standard uses the symmetric or unsymmetric solver is not supported in Abaqus/CAE.

Generating Substructures with Unsymmetric Reduced Damping Matrices

Usually, the reduced substructure operators (matrices) are symmetric, but the substructure stiffness and damping matrices can be unsymmetric for a number of special modeling cases. For example:

- When a coupled acoustic-structural substructure, generated from coupled or uncoupled modes, is generated from a model with damping specified on the acoustic domain, the substructure damping matrices are unsymmetric.
- The substructure stiffness matrix is unsymmetric if the substructure is generated from a model including sliding contact with friction. If the damping matrix is dependent on the stiffness matrix (for example, Rayleigh damping), the substructure damping matrix is unsymmetric.
- The substructure viscous damping matrix is unsymmetric if the substructure is generated from a rolling tire.
- The substructure viscous damping matrix is unsymmetric if the friction-induced contributions are included.

Unless requested explicitly in the substructure generation procedure, Abaqus/Standard generates a symmetric reduced viscous/structural damping matrix in a step using the symmetric solver and an unsymmetric reduced viscous/structural damping matrix in a step using the unsymmetric solver. To compute a substructure with unsymmetric viscous/structural damping matrices, you can do either of the following:

1. Generate the substructure using the unsymmetric solver. By default, Abaqus/Standard generates an unsymmetric instance of the reduced viscous/structural damping matrices.
2. Generate the substructure using the symmetric solver, and choose to generate a symmetric instance, an unsymmetric instance, or both a symmetric and an unsymmetric instance of the reduced viscous/structural damping matrices, as described in *Generating a Reduced Viscous Damping Matrix for a Substructure* and *Generating a Reduced Structural Damping Matrix for a Substructure*.

Option 2 allows you to take advantage of the performance of the symmetric solver while still being able to generate a substructure with unsymmetric viscous/structural damping matrices. The generated unsymmetric viscous/structural damping matrices can differ from those obtained using Option 1 if the model has any sources of unsymmetry in stiffness (for example, the presence of sliding contact with friction).

Defining Substructure Load Cases for Subsequent Loading in an Analysis

The load cases defined during the generation of a substructure and activated at the usage level are the equivalent of the elemental loading types available for the regular elements in Abaqus. They can be made up of any combination of loadings (distributed loads, concentrated nodal loads, thermal expansion, and load cases defined for any substructures that might be used as part of the definition of this substructure).

The load cases are needed so that, when the substructure is subsequently used in a model, the consistent loads on the retained degrees of freedom need be scaled only by the appropriate magnitudes of the particular loads applied: it is not necessary to go inside the substructure and repeat the basic element calculations to distribute the loads.

Each such load case can be applied when the substructure is used by associating it with an amplitude/time curve and a magnitude (*Amplitude Curves*). When a substructure is used, the substructure load case loadings that were created when the substructure was generated are the only loads that can be used in that substructure. Except for gravity loading, when using the substructure, you cannot apply distributed loads, temperature loads, etc. to the elements that make up any substructure. These loads must be built into the substructure during its creation.

You can define multiple substructure load cases during the substructure generation to define different loadings for the substructure. Each load case is assigned a name that will be used when the load case is applied on the usage level.

You can use any combination of concentrated load, distributed load, substructure load, and temperature fields (*Concentrated Loads* and *Distributed Loads*) to define each load case.

You assign each basic loading a reference magnitude, which will then be scaled by the actual magnitude specified when the substructure load is applied. The reference magnitude assigned to each basic loading must be defined as the change in load or boundary condition from the base state, not the total of the base state plus the perturbation value. Initial conditions applied within the substructure generation are not included as part of a load case definition.

For temperature loads, the load vector for the substructure load case contains only the contributions due to thermal expansion (see *Computing Thermal Strains in Linear Perturbation Steps*). If temperature-dependent material properties are present, they are evaluated at the temperatures specified in the preloaded state. Consequently, to take into account nonzero initial temperature fields prescribed as initial conditions (*Initial Conditions*), it is necessary to preload the structure before creating the substructure. When using temperature loading in a substructure load case, the data cannot be read from a results file. The temperatures specified must be defined as the change in the temperatures from the base state.

Abaqus/Standard currently has a limitation when a substructure load case definition includes acoustic loading during a substructure generation procedure in which retained modes are specified: the contribution of the singular (constant pressure) acoustic modes (*Coupled Acoustic-Structural Analysis*) is not taken into account in the generated load case. Since the contribution of this mode is significant for low frequency response, the generated load case will inadequately represent the specified acoustic load in these cases. If there are no singular acoustic regions in the coupled acoustic-structure substructure, the acoustic loads are represented accurately.

It is important to distinguish the difference between a load case and a preload. Both are defined during substructure generation, but only the preloads are actually applied to the substructure on the generation level; load cases, defined on the generation level, can only be applied on the usage level, and they act on a preloaded base state if one has been specified. (Preloads were discussed earlier.)

In general analysis steps and perturbation steps substructure loads are treated in the same way as other loads, such as concentrated loads and distributed loads (*Concentrated Loads* and *Distributed Loads*). For example, if a general analysis step is followed by another general analysis step, the substructure loads will be retained in the second step with their magnitude equal to that at the end of the previous general analysis step, unless the substructure load is modified or removed. In a linear perturbation step the substructure load represents an incremental load.

If a substructure load is used to apply Coriolis loading in a direct-solution steady-state dynamic analysis, the unsymmetric load stiffness contribution is not taken into account.

Input File Usage: Use the following options:

**SUBSTRUCTURE LOAD CASE*, NAME=*name*
**CLOAD* and/or
**DLOAD* and/or

*DSLOAD and/or
*TEMPERATURE

Use the following option for substructure load cases that include distributed nodal pressure loading:

*SUBSTRUCTURE LOAD CASE, NAME=*name*, GENERATE
*DSLOAD

The load case defined via the *SUBSTRUCTURE LOAD CASE option ends when an option other than *CLOAD, *DLOAD, *DSLOAD, or *TEMPERATURE is encountered. The load definitions can be specified in any order.

Abaqus/CAE Usage: Use the following option to define a substructure load case and the loads included in it:

Load module: **Create Load Case**: click the **Add** button: select loads

Defining Boundary Conditions

All boundary conditions to be built into the substructure matrices must be specified using a boundary condition definition. These cannot be part of a substructure load case specification. Once a kinematic boundary condition is specified on a particular nodal degree of freedom, it is built into the substructure matrices, is in effect for all load cases, and cannot be removed (or redefined at the usage level). The boundary conditions specified as part of the preloading history are built into the substructure matrices.

If there is any doubt whether a restraint is permanent or not, it is better to make the degree of freedom a retained degree of freedom and not specify any restraint in the substructure definition. The restraint can then be included as needed in each analysis step.

Load Cases When the Substructure Is Used in Geometrically Nonlinear Analyses

All loads included in a substructure load case at the generation level and applied as a substructure load at the usage level are applied in a local system associated with the substructure. Since this system rotates with the substructure when large motions are present, these loads will rotate as well. As a consequence, you should be careful when using substructure load cases in geometrically nonlinear analyses to ensure that the loading is in the appropriate direction at the usage level. This situation is similar to rotating the substructure using a substructure property definition.

Gravity Loading

To apply gravity loading, density must be defined for at least some of the elements included in the substructure. A gravity load can be applied to a substructure in two different ways with two different interpretations. If a distributed load definition is used as a part of a substructure load case during substructure generation (as described in *Defining Substructure Load Cases for Subsequent Loading in an Analysis* above), the gravity loading becomes part of the substructure load case and, hence, rotates to follow the substructure's local system during usage (the local system may rotate by rotating the substructure via a substructure property definition or due to geometrically nonlinear response).

To define gravity loading that acts in a fixed global direction during usage, you can request that the substructure's gravity load vectors be calculated during substructure generation. In this case gravity loading should not be defined as part of a substructure load case. When the gravity load vectors are calculated, Abaqus/Standard generates a gravity load vector for each global direction (three for three-dimensional analyses and two for two-dimensional/axisymmetric analyses). At the usage level, a distributed load definition can be used (see *Gravity*

Loading) to specify gravity loading on the substructure that acts in a fixed global direction with the specified magnitude.

Input File Usage: Use the following option to calculate the substructure's gravity load vectors during substructure generation:

**SUBSTRUCTURE GENERATE*, GRAVITY LOAD=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute gravity load vectors**

Substructure Eigenvalue Problem

We define the substructure eigenvalue problem as the generalized eigenvalue problem for reduced substructure stiffness and mass matrices. The reduced stiffness matrix is always generated for Abaqus substructures. If a generated substructure has the reduced mass matrix, the substructure eigenvalue problem can be solved and the substructure eigenmodes can be extracted. The substructure eigenfrequencies provide useful information about the substructure dynamic properties. The substructure eigenmodes can be used to define the substructure modal damping at the substructure usage stage, and they are required for the flexible body generation. By default, the substructure eigenvalue problem is solved when it is possible (when the reduced mass matrix is available). If generation of the reduced mass matrix is not requested but generation of a flexible body from a substructure is performed, we solve the substructure eigenvalue problem; but instead of the conventional reduced mass matrix, we use a projection of the lumped mass matrix on the substructure modal subspace. The lumped mass matrix is created from the global mass matrix of the finite element model by the commonly used heuristic algorithm. If the substructure eigenvalue problem is solved, the obtained substructure eigenvalues and eigenfrequencies are printed in the data (`.dat`) file. If desired, you can disable the solve of the substructure eigenvalue problem.

Input File Usage: Use the following option to enable solving of the substructure eigenvalue problem:

**SUBSTRUCTURE GENERATE*, EIGENPROBLEM=YES (default)

Use the following option to disable solving of the substructure eigenvalue problem:

**SUBSTRUCTURE GENERATE*, EIGENPROBLEM=NO

Abaqus/CAE Usage: Disabling the substructure eigenvalue problem solution is not supported in Abaqus/CAE. When it is possible, the substructure eigenvalue problem is solved.

Adding Residual Modes to the Substructure Basis

You can add residual modes to the substructure basis to improve the high-frequency dynamic approximation capability of the substructure. The residual modes are computed in previous static perturbation analyses and as the load modes corresponding to the substructure load cases specified in the current substructure generation analysis.

Input File Usage: Use the following option to add residual modes to the substructure basis:

**SUBSTRUCTURE GENERATE*, NAME=*name*, RESIDUAL MODES

Abaqus/CAE Usage: Adding residual modes to the substructure basis is not supported in Abaqus/CAE.

Adding Residual Modes Using a Single Analysis

The residual modes added to the substructure basis include the modes corresponding to the substructure load cases you specify in the substructure generation step and the responses from the static perturbation steps (with residual modes requested) between the current step and the most recent frequency step.

Example: Using a Single Analysis

This example describes the workflow to add residual modes using a single analysis with four steps.

- The first step is the frequency step whose eigenmodes are selected to be added to the fourth step, which generates a substructure with residual modes added to the substructure basis.
- The second and third steps are static perturbation steps (with two and one load cases, respectively) that write the computed responses in *jobname*_RM_*2*.sim and *jobname*_RM_*3*.sim, respectively.
- The substructure generation step also includes two substructure load cases, and the modes corresponding to them are added to the substructure basis as two residual modes.
- As discussed earlier, the static perturbation responses from the steps between the substructure generation step and its most recent frequency step are added to the substructure basis as residual modes; that is, the responses stored in *jobname*_RM_*2*.sim and *jobname*_RM_*3*.sim will be added.

Therefore, in this example, a total of five residual modes are added to the substructure basis:

- Two residual modes from the static perturbation responses corresponding to the two load cases in the second step
- One residual mode from the static perturbation response corresponding to the third step
- Two residual modes corresponding to the two substructure load cases in the fourth step

```
** Frequency step
*STEP, PERTURBATION
*FREQUENCY

*END STEP
** Static step with two residual modes
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES
*LOAD CASE, NAME=load1

*END LOAD CASE
*LOAD CASE, NAME=load2

*END LOAD CASE
*END STEP
** Static step with one residual mode
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES

*END STEP
** Substructure generation procedure
*STEP
*SUBSTRUCTURE GENERATE, NAME=name, RESIDUAL MODES
*RETAINED NODAL DOFS

*SELECT EIGENMODES

*SUBSTRUCTURE LOAD CASE, NAME=sload1

*SUBSTRUCTURE LOAD CASE, NAME=sload2
```

```
*END STEP
```

Adding Residual Modes Using the Restart Capability

The eigenfrequency extraction and substructure generation analyses can be performed only in SMP mode. In the workflow described above (which uses a single analysis), the analysis can be run only in SMP mode. This process can be slow when you need to compute many residual modes in static perturbation steps. However, you can use the restart capability to compute the residual modes from static perturbation steps in DMP mode and run the frequency and substructure generation steps in SMP mode, as described in this section.

The workflow involves running the frequency, static perturbation (with residual modes requested) and substructure generation (with residual modes requested) procedures in different jobs and in the specified order.

1. The first job specifies the model definition and frequency step and runs in SMP mode. This job generates eigenmodes (and possibly some pseudo eigenmodes, as described in *Natural Frequency Extraction*) to use in the substructure generation procedure.
2. The second job restarts from the first job, consists of several static perturbation steps (as required and with residual modes requested), and runs in DMP mode. This job generates many `.sim` files with static perturbation responses in them, to use as residual modes in the substructure generation procedure.
3. The third job restarts from the first job, consists of a substructure generation step (with residual modes requested) and any substructure load case definitions, and runs in SMP mode. This job adds all the modes corresponding to the substructure load case definitions and the residual modes output by the static perturbation steps in the second job to the substructure basis.

You can use the substructure generated as specified in *Using Substructures*.

Example: Using the Restart Capability

This example describes the workflow to add residual modes using the restart capability and runs the same series of steps discussed in the example for the workflow using a single analysis.

The first job (*job1name*) defines the model and the frequency step. This job computes the eigenmodes (and possibly some pseudo eigenmodes) in the second step.

```
*RESTART, WRITE
** Frequency step
*STEP, PERTURBATION
*FREQUENCY

*END STEP
```

The second job (*job2name*) defines the static perturbation analyses with residual modes requested. This job restarts from the first job and outputs static perturbation responses in two files: *job1name*_RM_*2*`.sim` and *job1name*_RM_*3*`.sim`. The `.sim` files use the first job name; and contain the step numbers from the second job, which continue from the step numbers in the first job.

The second job runs in DMP mode. You can add these residual modes to the subsequent substructure generation analysis, although only the steps within 100 steps of the frequency step are considered for residual modes in the substructure generation step. For example, because the frequency step in the first job is the first step, only the step numbers between 2 and 101 are considered for the addition of residual modes to the substructure. The second job can also contain a direct steady-state dynamic step with retained degrees of freedom to define a frequency-based substructure (see *Generating Frequency-Based Substructures*) that can be added to the conventional substructure in the subsequent substructure generation step.

```
*RESTART, READ
** Static step with two residual modes
*STEP, PERTURBATION
```

```
*STATIC, RESIDUAL MODES
*LOAD CASE, NAME=load1

*END LOAD CASE
*LOAD CASE, NAME=load2

*END LOAD CASE
*END STEP
** Static step with one residual mode
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES

*END STEP
```

The third job (*job3name*) defines the substructure generation analysis, runs in SMP mode, and restarts from the first job. The substructure basis includes selected eigenmodes computed in the frequency procedure in the first job, residual modes resulting from the substructure load cases, and the static perturbation–based responses in *job1name*_RM_*2*.sim and *job1name*_RM_*3*.sim.

```
*RESTART, READ
** Substructure generation procedure
*STEP
*SUBSTRUCTURE GENERATE, NAME=name, RESIDUAL MODES
*RETAINED NODAL DOFS

*SELECT EIGENMODES

*SUBSTRUCTURE LOAD CASE, NAME=sload1

*SUBSTRUCTURE LOAD CASE, NAME=sload2

*END STEP
```

Modifying and Adding More Residual Modes Using the Restart Capability

You can replace the already computed and added residual modes to the substructure with minimal computational effort. You run the first job described in *Example: Using the Restart Capability* only once, and do not run the job again if you need to replace the residual modes computed in the static perturbation procedures in the second job. The example above has two static perturbation steps with residual modes output by both steps.

If you need to replace the residual modes output by the first step in the second job, run a new job that restarts from the first job (which contains the frequency step) as shown below.

You can run this job in DMP mode. Because the step number of the only static perturbation step is 2 (the first job already has one step), this job generates *job1name*_RM_*2*.sim and replaces the existing file. You run the third job in the previous example for the substructure generation analysis to use the newly generated .sim file for residual modes in the substructure basis. This generated substructure has the residual modes generated in the first step of the second job replaced.

```
*RESTART, READ
** Static step with two residual modes
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES
*LOAD CASE, NAME=load3

*END LOAD CASE
*LOAD CASE, NAME=load4
```

```
*END LOAD CASE
*LOAD CASE, NAME=load5

*END LOAD CASE
*END STEP
```

If you need to replace the residual modes output in the second step of the second job in *Example: Using the Restart Capability*, you can add some steps (you can also use dummy steps) in the new job before the static perturbation step whose output responses you want to use to replace with the existing ones in *job1name*_RM_*3*.sim as shown below.

You can use a dummy step as the first step in this job, whose only purpose is to increment the step number of the second step to step number 3. The generated *job1name*_RM_*3*.sim replaces the existing file. Abaqus issues a warning message in the *job2name*.msg file when such a replacement occurs. You run the substructure generation job again, which uses the newly generated residual modes.

```
*RESTART, READ
** Dummy step
*STEP, PERTURBATION
*STATIC
*END STEP
** Static step with one residual mode
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES

*END STEP
```

You can also add more residual modes by running a new job with many dummy steps so that a new .sim file with residual modes is generated that does not replace any existing files.

The job in the example below restarts from the first job with the frequency step and includes two dummy steps so that the other two steps have step numbers 4 and 5 and output new .sim files: *job1name*_RM_*4*.sim and *job1name*_RM_*5*.sim. You then run the substructure generate job again, restarting from the frequency step. The residual modes from the files *job1name*_RM_*2*.sim, *job1name*_RM_*3*.sim, *job1name*_RM_*4*.sim, and *job1name*_RM_*5*.simare added to the substructure basis along with any modes corresponding to substructure load cases.

You can always add more residual modes to the existing substructure, without rerunning the first job with the frequency step (and potentially other steps) using this workflow.

```
*RESTART, READ
** Dummy step
*STEP, PERTURBATION
*STATIC
*END STEP
** Dummy step
*STEP, PERTURBATION
*STATIC
*END STEP
** Static step with one residual mode
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES

*END STEP
** Static step with one residual mode
*STEP, PERTURBATION
*STATIC, RESIDUAL MODES
```

```
*END STEP
```

Checking Generated Substructure Matrices

In a substructure generation analysis, you can check the quality of the generated substructure stiffness and mass matrices. The matrix check generates six “artificial” rigid body modes and projects the substructure matrices onto the rigid body modal subspace. It is expected that the projected 6×6 stiffness matrix (also known as the rigid body energy matrix) is close to zero in the absence of the boundary conditions and constraints. The total inertia statistics for the model are extracted from the projected 6×6 rigid body mass matrix. You can specify the center of rotation for creating the artificial rotational rigid body modes and calculating the global inertia tensor.

In a substructure generation analysis, the condition number of the substructure stiffness and mass matrices is also calculated. Outliers of the diagonal elements of the substructure stiffness and mass matrices are identified, together with their node and degree-of-freedom labels. This allows you to identify the retained degrees of freedom that can cause poor numerical quality of the substructure.

You can modify the tolerance values to use for the matrix check. If you request the matrix quality check, the check results are printed in the data (`.dat`) file.

You can decide whether problems reported by the matrix check are treated as errors. By default, problems reported by the matrix check are intended as warnings.

Input File Usage: Use the following option to request the matrix check:

**MATRIX CHECK*

Use the following option to treat any problem reported by the matrix check as an error:

**MATRIX CHECK*, ERROR

Use the following option to specify the center of rotation of the coordinate frame to use for the matrix check:

**MATRIX CHECK*, REFERENCE NODE=*node_id*

Use the following option to modify the tolerance values to use for the matrix check:

**MATRIX CHECK*, TOLERANCE=ON

Abaqus/CAE Usage: Checking generated substructure matrices is not supported in Abaqus/CAE.

Generating a Flexible Body

Abaqus/Standard can generate a flexible body from a substructure. Abaqus/Standard supports generation of several flexible body types for several external flexible body dynamics solvers. The generated flexible body entities are stored in the substructure `.sim` file, and the postprocessing programs (translators) are available to convert an Abaqus substructure into the conventional flexible body representation of a particular external flexible body dynamics solver.

Input File Usage: Use the following option to generate flexible body entities for the ADAMS™ flexible body dynamics solver from MSC.Software Corporation:

**FLEXIBLE BODY*, TYPE=ADAMS

Use the following option to generate the flexible body entities for the AVL EXCITE™ flexible body dynamics solver from AVL LIST GmbH:

**FLEXIBLE BODY*, TYPE=EXCITE

Use the following option to generate a generic flexible body (default):

**FLEXIBLE BODY*, TYPE=GENERIC

Use the following option to generate the Standard Input Data representation of the flexible body:

**FLEXIBLE BODY*, TYPE=SID

Use the following option to generate flexible body entities for the Simpack flexible body dynamics solver:

**FLEXIBLE BODY*, TYPE=SIMPACK

Abaqus/CAE Usage: Generating a flexible body from a substructure is not supported in Abaqus/CAE.

Different Flexible Body Formulations

When it is applicable, you can generate different versions of the flexible body for the AVL EXCITE™ flexible body dynamics solver from AVL LIST GmbH or the flexible body for the ADAMS™ flexible body dynamics solver from MSC.Software Corporation. Generating the reduced version of the flexible body can significantly reduce the generation time for a large substructure. You should decide if the particular reduced version is applicable based on the engineering nature of the analysis.

Input File Usage: Use the following option to generate a particular version of the flexible body for the ADAMS™ flexible body dynamics solver from MSC.Software Corporation:

**FLEXIBLE BODY*, TYPE=ADAMS, FORMULATION=*formulation type*

Use the following option to generate a particular version of the flexible body for the AVL EXCITE™ flexible body dynamics solver from AVL LIST GmbH:

**FLEXIBLE BODY*, TYPE=EXCITE, FORMULATION=*formulation type*

Abaqus/CAE Usage: Generating a flexible body from a substructure is not supported in Abaqus/CAE.

Converting Substructures to a Flexible Body Format

Abaqus provides translators that can read data from a substructure SIM file and write data to a conventional input file for external flexible body dynamics solvers.

You can use the **abaqus tosimpack** translator to convert an Abaqus substructure to a flexible body in a format of the Simpack multibody dynamics solver (see *Translating an Abaqus Substructure to a Simpack Flexible Body*).

You can use the **abaqus toexcite** translator to convert an Abaqus substructure to a flexible body in a format of the AVL EXCITE™ multibody dynamics solver by AVL LIST GmbH (see *Translating an Abaqus Substructure to an EXCITE Flexible Body*).

You can use the **abaqus adams** translator to convert an Abaqus substructure to a flexible body in a format of the ADAMS™ multibody dynamics solver from MSC.Software Corporation (see *Translating an Abaqus Substructure to an MSC.Adams Flexible Body*).

By default, the **abaqus** execution procedure executes the **tosimpack** or **toexcite** translators with default arguments on completion of an Abaqus/Standard flexible body generation analysis set to generate a flexible body in the Simpack or AVL EXCITE™ format, respectively. You can overwrite this default by specifying the **abaqus** execution procedure argument **noFlexBody**.

Writing the Recovery Matrix, Reduced Stiffness Matrix, Mass Matrix, Load Case Vectors, and Gravity Vectors to a File

You can write a substructure's recovery matrix, reduced stiffness matrix, mass matrix, and load case vectors to a file. This output is useful when the substructure is to be used in another program.

The output records can be written either to the Abaqus/Standard results file, to a user-defined file, or to the output database file (see below). In each case you must specify which matrices/vectors to output: the mass matrix, the recovery matrix, the load case vectors, the stiffness matrix, and/or the gravity load vectors. By default, no output will be generated.

Repeat the substructure matrix output request in the substructure generation file of each substructure for which the substructure matrix output is required.

If substructure load case vector output is requested for a preloaded substructure, the output will contain a record with a load case number that is equal to zero. This load vector contains the forces that were necessary to equilibrate any stresses that were generated during the previous steps.

Input File Usage: **SUBSTRUCTURE MATRIX OUTPUT*, MASS=YES, RECOVERY MATRIX=YES, SLOAD=YES, STIFFNESS=YES, GRAVITY LOAD=YES

Abaqus/CAE Usage: Writing a substructure's recovery matrix, reduced stiffness matrix, mass matrix, load case vectors, and gravity vectors to a file is not supported in Abaqus/CAE.

Writing the Records to the Abaqus/Standard Results File

By default, the requested matrices are written to the Abaqus/Standard results file corresponding to the substructure generation input file name. The record formats for the results file are described in *Results File*. The file can be written in either binary or ASCII format (*About Output*).

Input File Usage: **SUBSTRUCTURE MATRIX OUTPUT*, OUTPUT FILE=RESULTS FILE

Abaqus/CAE Usage: Writing a substructure's recovery matrix, reduced stiffness matrix, mass matrix, load case vectors, and gravity vectors to a file is not supported in Abaqus/CAE.

Writing the Records to a User-Defined File

You can specify the name of the file (without an extension) to which the data will be written. The records are written to be compatible with a linear user-defined element. The record formats are described in *User-Defined Elements*. An `.mtx` extension will be added to the file name specified.

Input File Usage: **SUBSTRUCTURE MATRIX OUTPUT*,
OUTPUT FILE=USER DEFINED, FILE NAME=*file_name*

Abaqus/CAE Usage: Writing a substructure's recovery matrix, reduced stiffness matrix, mass matrix, load case vectors, and gravity vectors to a file is not supported in Abaqus/CAE.

Generating Frequency-Based Substructures

Products: Abaqus/Standard

References:

- *Using Substructures*
- **RETAINED NODAL DOFS*
- **STEADY STATE DYNAMICS*
- **SELECT EIGENMODES*
- **RESTART*

Overview

This section describes how to define individual frequency-based substructures.

Frequency-based substructures differ from conventional substructures (see *Generating substructures*) in that they use stiffness, inertial, damping, and frequency information to generate a single complex, condensed operator at every specified frequency point.

You can use frequency-based substructures only in direct steady-state dynamic analyses (see *Direct-Solution Steady-State Dynamic Analysis*).

For information on how substructures are used in a model, see *Using Substructures*.

You define a frequency-based substructure using a combination of a direct steady-state dynamic analysis and a substructure generation procedure. The direct steady-state dynamic analysis coupled with retained external degrees of freedom generates the operators corresponding to the frequency-based substructure at the user-specified frequencies. The substructure generation procedure stores both the frequency-based substructure operators and the conventional substructure operators on the merged substructure database. One or the other set of operators is used depending on the analysis type and substructure property settings.

All general behaviors related to substructure generation and usage apply to frequency-based substructures. This section discusses details that are pertinent to frequency-based substructure generation.

Frequency-based substructures work in direct steady-state dynamic analyses in the same way that conventional substructures work in static analyses. Conventional substructures provide exact (up to round-off error) representation of the model in static analyses, while frequency-based substructures provide exact representation of the model in direct steady-state dynamic analyses. When the frequency-based substructure is used at the same frequencies as those used for generation, the dynamic response is exact (up to round-off error). The response at frequencies that do not match those used for generation can be approximate (see *Using Substructures*).

To define frequency-based substructures, do the following:

1. Invoke the direct steady-state dynamic procedure.
   a. Define the frequency points at which to generate a frequency-based substructure.
   b. Define the nodes and degrees of freedom to retain as external degrees of freedom when using the frequency-based substructure.
2. Invoke the substructure generation procedure.
   a. Define the nodes and degrees of freedom to retain as external degrees of freedom when using the frequency-based substructure. These degrees of freedom must be the same as those specified in the direct steady-state dynamic procedure.

b. Define the rest of the substructure generation procedure (see *Generating substructures*).

You can define frequency-based substructures using a single analysis or using the restart capability. Because the substructure generation procedure runs only in shared memory parallel (SMP) mode, the performance can be limited when the direct steady-state dynamic analysis and substructure generation procedure are run in a single analysis. You can use the restart capability to run two separate jobs and allow the direct steady-state dynamic analysis to run in distributed memory parallel (DMP) mode to reduce the computational expense, particularly for large models.

Generating Frequency-Based Substructures in a Single Analysis

You define the direct steady-state dynamic analysis as described in *Direct-Solution Steady-State Dynamic Analysis*. You specify the frequencies at which to generate the frequency-based substructure. You define the retained degrees of freedom as described in *Defining the Retained Nodal Degrees of Freedom*. You should limit the number of retained degrees of freedom that you specify based on the size of the model because every retained degree of freedom is associated with an internally created right-hand-side vector. Specifying many retained degrees of freedom can affect the computational time and memory consumption in the direct steady-state dynamic analysis.

By default, all previously applied boundary conditions are removed in a direct steady-state dynamic analysis coupled with retained degrees of freedom. If you want to retain the boundary conditions from a previous loading step, you must respecify those boundary conditions in the direct steady-state dynamic analysis. You must ensure that no boundary conditions are specified on the retained degrees of freedom.

When you define a direct steady-state dynamic analysis coupled with retained degrees of freedom, no additional loads or load cases are allowed.

The direct steady-state dynamic analysis coupled with retained degrees of freedom writes the generated frequency-based substructure operators in the *jobname*`_FBS.sim` file in the output directory. The substructure generation procedure that immediately follows the direct steady-state dynamic analysis stores the frequency-based substructure operators and the conventional substructure operators in the *jobname*`_Z`*n*`.sim` file written by the substructure generation procedure (see *Generating substructures*).

Input File Usage: Use the following options to define a frequency-based substructure using a direct steady-state dynamic analysis coupled with retained degrees of freedom and a substructure generation step with the same retained degrees of freedom in a single analysis:

**STEP*
**STEADY STATE DYNAMICS*, DIRECT

**RETAINED NODAL DOFS*

**BOUNDARY*, OP=NEW

**END STEP*
** Substructure generation procedure
**STEP*
**SUBSTRUCTURE GENERATE*
**RETAINED NODAL DOFS*

**END STEP*

Generating Frequency-Based Substructures Using the Restart Capability

You can run the direct steady-state dynamic analysis coupled with retained degrees of freedom and the conventional substructure generation procedure in separate jobs using the Abaqus restart capability (see *Restarting an Analysis*). The first job includes the direct steady-state dynamic analysis coupled with retained degrees of freedom, as described in *Generating Frequency-Based Substructures in a Single Analysis*. In this job, you can apply preloads to the substructure before the direct steady-state dynamic step. This job can run in DMP mode. This job writes the frequency-based substructure operators to the *job1name*_FBS.sim file.

The substructure generation procedure is then run in a restart job that creates a merged substructure database that includes the frequency-based substructure operators generated in the previous job. Optionally, a frequency step to obtain eigenmodes to add to the conventional substructure can precede the substructure generation procedure. The restart job runs only in SMP mode. The restart job writes the frequency-based substructure operators and the conventional substructure operators together in the *job2name*_Z*n*.sim file.

The generated substructure includes both the frequency-based substructure and the conventional substructure and is used in a similar way as the conventional substructure (see *Using substructures*).

Example 1: Generating Frequency-Based Substructures Using the Restart Capability

This example shows how to use the restart capability to generate frequency-based substructures. The first job includes the direct steady-state dynamic analysis coupled with retained degrees of freedom.

```
*RESTART, WRITE
*STEP
*STEADY STATE DYNAMICS, DIRECT

*RETAINED NODAL DOFS

*END STEP
```

The second job includes the substructure generation procedure.

```
*RESTART, READ
*STEP
*SUBSTRUCTURE GENERATE, NAME=name
*RETAINED NODAL DOFS

*END STEP
```

Example 2: Generating Frequency-Based Substructures Using the Restart Capability

You can also generate frequency-based substructures using a three-job workflow.

1. The first job (*job1name*) specifies the model definition and frequency step and runs in SMP mode.
2. The second job (*job2name*) restarts from the first job, consists of a direct steady-state dynamic step with retained degrees of freedom, and runs in DMP mode. The second job writes the frequency-based substructure operators to the *job1name*_FBS.sim (the name of the first job is used for file output).
3. The third job (*job3name*) consists of a substructure generation step with the same retained degrees of freedom as those specified in the direct steady-state dynamic step in the second job and runs in SMP mode.

The example below demonstrates this three-job workflow.

The first job includes the model definition and the frequency step.

```
*RESTART, WRITE
*STEP
*FREQUENCY
```

```
*END STEP
```

The second job includes the direct steady-state dynamic step with retained degrees of freedom.

```
*RESTART, READ
*STEP
*STEADY STATE DYNAMICS, DIRECT

*RETAINED NODAL DOFS

*END STEP
```

The third job includes the substructure generation step with the same retained degrees of freedom as in the direct steady-state dynamic step in the second job.

```
*RESTART, READ
*STEP
*SUBSTRUCTURE GENERATE, NAME=name
*RETAINED NODAL DOFS

*END STEP
```

Limitations

- Frequency-based substructures do not support substructure loads.
- The boundary conditions in the direct steady-state dynamic procedure coupled with retained degrees of freedom are internally assumed to be defined using OP=NEW.
- The selected eigenmodes improve the quality only of the conventional substructure and not the frequency-based substructure.

Submodeling

In this section:

- *About Submodeling*
- *General Submodeling*
- *Node-Based Submodeling*
- *Surface-Based Submodeling*

About Submodeling

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Node-Based Submodeling*
- *Surface-Based Submodeling*
- *Output to the Output Database*
- **SUBMODEL*
- *Submodeling*

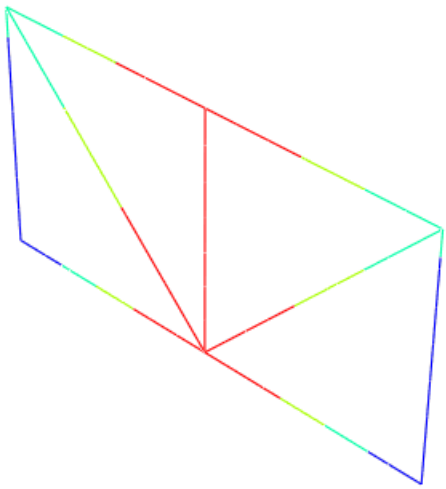
Overview

The submodeling technique:

- is used to study a local part of a model with a refined mesh based on interpolation of the solution from an initial (undeformed), relatively coarse, global model;
- is most useful when it is necessary to obtain an accurate, detailed solution in a local region and the detailed modeling of that local region has negligible effect on the overall solution;
- can be used to drive a local part of the model by nodal results, such as displacements (*General Submodeling*, *Node-Based Submodeling*), or by the element stress results (*Surface-Based Submodeling*) from the global mesh;
- can be used to analyze an acoustic model driven by displacements from a structural, global model when the acoustic fluid has negligible effect on the structural solution;
- can be used for the analysis of a structure driven by acoustic pressures from an acoustic or coupled acoustic-structural, global model;
- can be used to drive a surface representing a beam of a submodel driven by the displacement and rotations of a structural beam element in the global model;
- can use a combination of Abaqus/Explicit and Abaqus/Standard procedures;
- can use a combination of linear and nonlinear procedures; and
- cannot be used to drive a time-domain analysis with the results from a frequency-domain analysis (or vice versa).

Introduction

The submodeling technique allows you to follow up a simulation with a detailed study of one or more local regions of interest, with the results from the original simulation providing the boundary conditions at submodel "cuts." The original model is referred to as the “global” model, and a subsequent detailed model is referred to as a “local” model or a “submodel.” You identify the regions to investigate with submodeling during postprocessing of the global model.



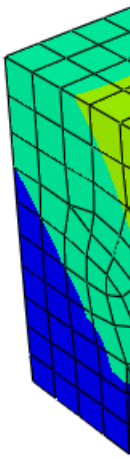


Figure 1: Example of a global model involving beam elements and a submodel involving shell elements.

Figure 1 shows a global model of a truss system involving beam elements and a submodel involving refined shell elements in the vicinity of a joint at the left top end of the truss. In this example, displacements and rotations at three "cut" boundaries of the submodel are imposed by results from the global model at these locations.

Submodeling is often significantly more efficient than using a globally refined model. The accuracy of submodel results relies on sufficient accuracy of the global results that drive results at the submodel boundary “cuts.” Submodeling leverages Saint-Venant’s principle that the difference between the effects of two different but statically equivalent loads becomes very small at sufficiently large distances from the loading. Judgment is required to position the submodel cuts far enough away from locations where you desire greater accuracy. It is good practice to confirm accuracy of the submodel results with refined global meshes for some representative cases. Occasionally, changes to modeling practices and accounting for more details within a localized region might influence a larger region than expected. Material properties are allowed to differ between the global model and the submodel; however, this might lead to inaccuracies if effects on solution variables extend to the submodel boundaries. For further discussion of submodeling accuracy, see *Accuracy*.

Submodeling Techniques

For an example application of the submodeling technique, see *Shell-to-solid submodeling and shell-to-solid coupling of a pipe joint*. Submodeling can be applied quite generally in Abaqus. Both the global model and the submodel can have nonlinear response. The material response defined for the submodel might be different from that defined for the global model.

Abaqus provides the following submodeling interfaces:

- general submodeling (see *General Submodeling*),
- node-based submodeling (see *Node-Based Submodeling*), and
- surface-based submodeling (see *Surface-Based Submodeling*).

Together these interfaces cover many submodeling scenarios. However, they differ in applicability depending on element type combinations in the global model and the submodel, what types of conditions are imposed on submodel cuts, whether submodel cuts are specified as surfaces or node sets, and whether they are used in Abaqus/Standard or Abaqus/Explicit.

You cannot use general submodeling with other submodeling types in the same model. You can use node-based submodeling and surface-based submodeling together.

Table 1 summarizes the applicability of general submodeling and node-based submodeling for driving displacements and, sometimes, rotations at submodel cuts for common combinations of three-dimensional element types in the global model and submodel including membrane elements. The example shown in *Figure 1* uses general submodeling. You can use general submodeling only in Abaqus/Standard; whereas you can use node-based submodeling in both Abaqus/Standard and Abaqus/Explicit. You can use surface-based submodeling only in Abaqus/Standard.

Table 1: Applicability of general submodeling (GS) and node-based submodeling (NBS) for driving submodel displacement and rotations for common three-dimensional element types.

		Submodel element type		
		Beam	**Shell**	**Solid**
Global element type	Beam	GS	GS	GS
	Shell		GS or NBS	GS or NBS
	Solid			GS or NBS

Node-based submodeling can also be used to drive:

- Displacements at submodel cuts from global results for like element types in the global model and the submodel: cylindrical elements, gasket elements, and surface elements.
- Nodal forces of solid elements at submodel cuts based on acoustic pressures determined in a global acoustic model.

Surface-based submodeling can be used to drive nodal forces at submodel cuts based on stress fields from a global model if the global and submodel have axisymmetric, two-dimensional, or three-dimensional solid elements. Surface-based submodeling does not impose kinematic constraints on cut surfaces; therefore, you might need to add minimal boundary conditions to suppress rigid-body modes.

Both general submodeling and node-based submodeling allow fundamentally different element types in the global model and submodel in the vicinity of a submodel cut surface. You do not specify what combination of global and submodel element types are associated with a particular submodel cut for general submodeling; however, you do need to specify if a particular node-based submodel definition is associated with shell-to-solid or acoustic-to-structure submodeling as discussed in *Specifying the Driven Nodes in the Submodel*.

Performing a Submodeling Analysis

A submodeling analysis consists of:

- running a global analysis and saving the results in the vicinity of the submodel boundary;
- defining the total set of driven nodes or driven surfaces in the submodel;
- defining the time variation of the driven variables in the submodel analysis by specifying the actual nodes and degrees of freedom or element-based surfaces to be driven in each step; and
- running the submodel analysis using the "driven variables" to drive the solution.

Linking the Global Model and the Submodel

The submodel is run as a separate analysis from the global analysis. The only link between the submodel and the global model is the transfer of the time-dependent values of variables saved in the global analysis to the relevant boundary nodes or to the relevant boundary surfaces of the submodel. The location where the results from the global model are saved depends on the technique used:

- Node-based using the submodel interface: results (`.fil`) file, output database (`.odb`) file, or SIM database (`.sim`) file
- Node-based using the field import interface: SIM database (`.sim`) file
- Surface-based (stress) technique: output database (`.odb`) file or SIM database (`.sim`) file
- General submodel: SIM database (`.sim`) file

The transfer is achieved by then reading these results into the submodel analysis. You should provide the name of the global model along with the extension using the **globalmodel** parameter on the execution procedure (see *Abaqus/Standard and Abaqus/Explicit Execution*) to indicate the type of the database to be used. The type of database used to save results from the global model can be specified on the execution procedure for the global model by providing a value for the **resultsformat** option (see *Additional Options Available for the Analysis Module*). If you do not provide the extension, Abaqus searches the execution directory in the following order:

1. the results file
2. the SIM database file
3. the output database file

If the global model is defined in terms of an assembly of part instances, the part (`.prt`) file from the global model is required for the submodel analysis.

Since the submodel is a separate analysis, submodeling can be used to any number of levels; a submodel can be used as the global model for a subsequent submodel.

Referring to the Global Model Results from the Submodel Analysis

You must define the source of the global solution results and provide the name of the output database file (in ODB or SIM format); the file extension is optional. If you omit the file extension, Abaqus uses the ODB output database file or the SIM database file, in that order.

Input File Usage: **abaqusjob=***submodel_input_file* **globalmodel=** *global_output_database* or *sim_database_file*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*submodel***: **Submodel**: **Read data from job**: *global_output_database*

Reading data from a SIM database file is not supported in Abaqus/CAE.

Accuracy

The global model in a submodeling analysis must define the submodel boundary response with sufficient accuracy. It is your responsibility to ensure that any particular use of the submodeling technique provides physically meaningful results. In general, the solution at the boundary of the submodel must not be altered significantly by the different local modeling. There is no built-in check of this criterion in Abaqus; it is a matter of judgment on your part. In general, accuracy can be checked by comparing contour plots of important variables near the boundaries of the submodeled region.

Minimizing File Sizes

The size of the results file or the output database can be minimized for a submodeling analysis by requesting output for only those global nodes and global elements that are used to drive the submodel. To determine which global nodes and/or elements are used to drive the submodel, do the following:

1. Run a data check analysis on the global model with any combination of results file or output database file output requests. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus (*Abaqus/Standard and Abaqus/Explicit Execution*).
2. Run a data check analysis on the submodel.

A listing of the global nodes and/or elements that will be used to drive the submodel is output to the data file during the submodeling data check analysis.

Frequency of Output

Pay special attention to the frequency at which you request output in the global model (see *Output to the Data and Results Files* and *Output to the Output Database*). It is possible to define the results file output or nodal and element output to the output database file such that the information is written at different frequencies for different nodes and elements, although that should not be done for nodes and elements involved in the interpolation to define values at driven variables since Abaqus will take values at the coarsest frequency only. To avoid this problem, write the nodal and elemental output to the output database or the results file using the same frequency for all nodes and elements involved in the interpolation and choose a frequency that will allow the history in the submodel to be reproduced accurately.

Input File Usage: To control the output frequency to the Abaqus/Standard results file, use the following option:

**NODE FILE*, FREQUENCY

To control the output frequency to the Abaqus/Explicit results file, use the following option:

**FILE OUTPUT*, NUMBER INTERVAL

To control the output frequency to the output database, use the following option:

**OUTPUT*, FIELD, FREQUENCY

Abaqus/CAE Usage: Step module: **Output->Field Output Requests->Create**: **Frequency**

Material Options

Any of the material models described in *Abaqus Materials Guide* can be used in the global and submodel analyses. The material response defined for the submodel might be different from that defined for the global model.

Elements

The dimensionality of the submodel must be the same as that of the global model: both models must be either two-dimensional or three-dimensional. The following limitations apply:

- The boundary nodes of the submodel must lie within regions of the global model where Abaqus is able to perform spatial interpolation to define the values of the driven variables. Therefore, they must lie within (or, as allowed by the exterior tolerance, near to) two- or three-dimensional geometrically defined elements in the global model. Such geometrically defined elements are:
  - first- or second-order triangles or quadrilaterals in two dimensions;
  - first- or second-order triangular or quadrilateral shells; and
  - first- or second-order tetrahedra, wedges, or bricks in three dimensions.
- The boundary nodes cannot lie in regions of the global model where there are only one-dimensional elements (beams, trusses, links, axisymmetric shells) since Abaqus does not provide the necessary interpolation of results for such elements.
- The boundary nodes cannot lie in regions of the global model where there are only user elements, substructures, springs, dashpots, cohesive elements, etc. since those element types do not allow for geometric interpolation.
- The boundary nodes cannot lie in regions of the global model where there are only axisymmetric solid elements with nonlinear, asymmetric deformation (CAXA elements). The submodeling capability is currently not supported for these elements.
- The reference node associated with generalized plane strain elements (CPEG) cannot be used as a driven boundary node in a submodeling analysis.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers*).

As described above, nodal output requests to the results file or output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

General Submodeling

Products: Abaqus/Standard

References:

- **SUBMODEL CONDITIONS*
- **SUBMODEL CUT*
- **SUBMODEL TIME SYNCHRONIZATION*
- **SUBMODEL LINEAR LOAD CASE REFERENCE*

Overview

General submodeling imposes the displacements, rotations, temperatures, or pore pressure obtained from the beam, solid, or shell elements in the global model onto a surface defined in the submodel.
The surface in the submodel is meshed with either beam, solid, or shell elements.

Introduction

General submodeling enables a strategy to use beam, solid, and shell elements in a global model to obtain an approximate solution efficiently followed by a submodel with beam, solid, or shell elements to study a region in more detail. Results from the global model impose conditions on cut surfaces of the submodel.

General Submodeling

Element Types Supported

General submodeling uses beam elements, solid elements, or shell elements in the submodel.

Procedures Supported

Both the global model and the submodel can have nonlinear response and can be analyzed for any sequence of analysis procedures. These procedures do not have to be the same for both models. For example, the linear dynamic response of the global model can be used to drive the static, nonlinear response of the submodel on the grounds that the submodel is too small for dynamic effects to be significant in that local region. You cannot use general submodeling in Abaqus/Explicit analyses.

Your submodel cannot refer to a global model step that includes multiple load cases (see *Multiple Linear Load Case Analysis*). You must perform the global analysis with a single load definition for the step that will drive the submodel.

Invoking General Submodeling

Creating and executing a submodel in the context of general submodeling involves the following:

- Specifying the named element-based surfaces that will be associated with submodel cuts as model data.
- Using the named surfaces to define submodel cuts as part of the model data .
- Specifying data that are common to all submodel cuts, such as the associated step and increment of the global model, as part of the submodel conditions definition for a step.
- Specifying the name of the associated global model with the **globalmodel** parameter on the Abaqus command line.

Input File Usage: Use both of the following options:

**SUBMODEL CUT*, LOCAL SURFACE=*element_based_surface*

**SUBMODEL CONDITIONS*

Imposed Conditions at a Submodel Cut

General submodeling imposes boundary conditions on primary degrees of freedom of the nodes on the cut surface either directly or by defining a distributed coupling element on the nodes of the cut surface. The following primary degrees of freedom can be imposed: displacement, rotation, temperature, pore pressure, and acoustic pressure. When the distributed coupling is defined, Abaqus/Standard imposes the boundary conditions on the reference node.

When you define the submodel cut surface on a beam cross-section or shell thickness, Abaqus/Standard internally generates a distributing coupling with the nodes on the cut surfaces acting as cloud nodes. *Figure 1* shows superimposed global and local models for beam-to-solid and beam-to-shell submodeling examples. These distributing couplings are represented with lines from the reference node to each of the cloud nodes at the four submodel cuts in *Figure 1*. Reference nodes for these distributing couplings are located at the intersection of the plane of the cut surface and the beam element reference line in the global model. Abaqus/Standard obtains the values of the imposed degrees of freedom at the reference node in the submodel by interpolation from the global model solution for the beam element at the location of the intersection. The distributing coupling causes the average displacement and rotation of the cut surface to match that of the reference node, without imposing rigidity. For example, the distributing coupling does not prevent warping of the cut surface during the submodel simulation.

Figure 1: Superimposed global and local models for beam-to-solid (left) and beam-to-shell (right) submodeling.

Figure 2 shows the superimposed global model and submodel for a case involving shell-to-shell and shell-to-solid submodeling together at different submodel cuts of one submodel. The coupling method used to impose conditions from the global model on a submodel for shell-to-shell submodeling involves interpolation from the global model results. *Figure 3* shows the coupling method for shell-to-solid submodeling using internally generated distributing couplings with cloud nodes on the cut surface of the submodel. The automated algorithm determines the number of reference nodes and their positions on the submodel interface. Displacements and rotations at reference nodes of the distributing couplings are interpolated from shell nodes of the global model. Each of the driven solid nodes belongs to a single distributing coupling.

Figure 2: Superimposed global and submodel for a global model with shell elements and a submodel with a combination of shell and solid elements.

Figure 3: Representation of automatically generated couplings to enforce shell-to-solid submodeling conditions on a solid surface submodel cut.

When you define the cut surface on the edges of shell elements and you impose the boundary conditions from shell elements in the global model using interpolation, you do not need distributing coupling elements.

Modifying the Imposed Conditions

By default, Abaqus/Standard imposes all primary degrees of freedoms that are common to both the global model and submodel on all cut surfaces. You can specify the degrees of freedoms to be imposed and the degrees of freedom to be excluded for each of the cut surfaces.

Input File Usage: Use following data lines:

**SUBMODEL CONDITIONS*
cut surface name, *First dof*, *Last dof*, P *or* N, *First dof*, *Last dof*, P, *or* N
cut surface name, *First dof*, *Last dof*, P *or* N
...

If you omit the name of the cut surface, the specification refers to all cut surfaces.

Enter the letter P to impose the primary degrees of freedom and the letter N to exclude the primary degrees of freedom.

Use a separate data line for each cut surface to modify the imposed conditions. The last specification takes precedence over any earlier specification.

Consistent Configuration and Orientation

Accurate submodel behavior for submodeling from a beam element in the global model requires consistency of the beam cross-section configuration in the global model and submodel. Initial nodal positions of the submodel must account for:

- Beam section offset in the global model.
- The orientation of principal beam axes in the global model.
- Beam section dimensions and thicknesses.

Referring to the Step and Increment of the Global Model

Step histories often match between a submodel and the corresponding global model. In these cases and some other cases with differences in step histories (as explained in *Figure 4* and *Figure 5*), the following default conventions for step synchronization (that is, associating the steps of a submodel with a global model) are appropriate:

- The *i*th general step of a submodel is driven by the *i*th general step of the associated global model.
- The *j*th perturbation step associated with a base state determined by the *i*th general step of a submodel is driven by the *j*th perturbation step determined by the *i*th general step of the associated global model.

In the example shown in *Figure 4*, the global model has an extra perturbation step after the second general step compared to the submodel. In this case, by default, the extra perturbation step is not used to drive the submodel conditions. Default step synchronization is reasonable for this example. However, if, for example, you intended to have Step 3 of the submodel driven by Step 4 of the global model, you could override the default.

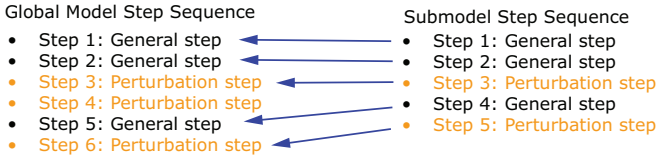


Figure 4: Example of default time synchronization referencing from submodel steps to global model steps that drive submodel conditions.

Figure 5 shows a similar example, but with an extra perturbation step in the submodel relative to the global model. In this case Abaqus issues an error message if you are using the default time synchronization because the global model only has one perturbation step after the second global step. For example, you could specify that Step 4 of the submodel (which is a perturbation step) should be driven by perturbation results from Step 5 of the global model, even though Step 5 of the global model uses a different base state. Such an association might be questionable, but it is not disallowed.

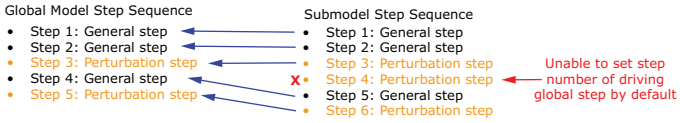


Figure 5: Example with default time synchronization unable to determine the default driving global step for one step of the submodel.

Figure 6 shows two examples of nondefault time synchronization references for submodeling. Both models consider the same four general steps, and default time synchronization reasonably applies to these general steps. Step 3 of the submodel is a perturbation step that the user has explicitly specified should be driven by incremental changes determined in the first increment of Step 3 of the global model.

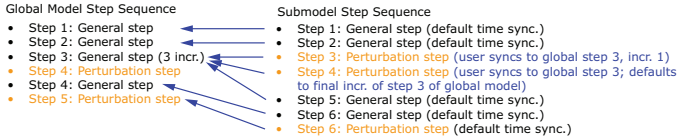


Figure 6: Example with default time synchronization used for four general steps of a submodel and nondefault synchronization specified for two of three perturbation steps of a submodel.

Input File Usage: Use following option.

**SUBMODEL TIME SYNCHRONIZATION*, GLOBAL STEP=*step_number*, INC=*increment_number*

Amplitude Scaling Between the Global Model and Submodel

Amplitude scaling enables you to scale the results of the global model that are used to drive conditions at submodel cuts. For example, an amplitude scale factor of 2.0 doubles the driven conditions at a submodel cut. The default scale factor is 1.0.

Input File Usage: Use following option.

**SUBMODEL TIME SYNCHRONIZATION*, AMPLITUDE SCALE=*value*

Time Scaling Between the Global Model and Submodel

Time periods of corresponding steps of a submodel and global model might differ. Time scaling occurs by default, such that the variations of driving conditions are based on fractional step completion of the global step and corresponding submodel step.

You can optionally deactivate time scaling. For example, if a global step has a period of 1.0 seconds and the corresponding submodel step has a period of 2.0 seconds, the time histories of conditions driving the submodel will match for the first second but will remain constant for the last half of the submodel step. This approach might be appropriate for a global model step with purely static behavior (which does not involve a physical time period) and a corresponding submodel step with transient static response involving time-dependent material behavior (creep, swelling, and viscoelasticity).

Input File Usage: Use following option to deactivate time scaling:

**SUBMODEL TIME SYNCHRONIZATION*, TIME SCALE=OFF

Controls Associated with Identifying the Global Element Associated with a Cut Surface

A search algorithm identifies global elements associated with each local cut surface. The default search algorithm is usually sufficient. Optionally, you can specify a global element set as input to this algorithm to limit which global elements are considered as candidates. If a cut surface of the local model corresponds to the location of a node shared by multiple beam elements in the global model, specifying one of these elements in a global element set removes uncertainty about which global element is chosen to drive the submodel conditions at that cut. Similarly, you can define tolerances to include or exclude entities that lie outside the boundaries of the submodel. You can specify the tolerances either as a percentage of the average element size or as absolute values.

Input File Usage: Use following option.

**SUBMODEL CUT*, ABSOLUTE EXTERIOR TOLERANCE=*value*, EXTERIOR TOLERANCE=*value*, GLOBAL ELSET=*element_set_name*

Using Multiple Linear Load Cases in a Submodel

Multiple linear load case functionality (see *Multiple Linear Load Case Analysis*) can be used in the context of general submodeling, which reduces memory and execution time compared to defining separate perturbation steps. Load cases can exist both in the global model analysis and in the submodel analysis. If the global step being referenced in a submodel analysis has multiple linear load cases, you must specify the name of the load case in the global model.

There are two methods that you can use for a submodel to refer to linear load cases of a global model, as follows:

- The first method allows multiple linear load cases to be considered in a single submodel step. In this case, you must assign names to the linear load cases in the context of the submodel step.

  This method is the most efficient way to consider multiple load cases associated with a common base state.
- The second method allows only one linear load case to be referenced per submodel step. This method does not require you to assign a name to the linear load case in the context of the submodel step.

  However, this method increases simulation time compared to the first method because some computations are repeated in each step.

Input File Usage: Use the following option in the global model:

**LOAD CASE*, NAME=*GlobalLoadCaseName*

Use the following option in the submodel:

**SUBMODEL LINEAR LOAD CASE REFERENCE*, GLOBAL LOAD CASE=*GlobalLoadCaseName*

Example:

Consider a step of a global model with three linear load cases:

```
*STEP
*LOAD CASE, NAME=GlobalLoadCaseName1
*CLOAD
*END LOAD CASE
*LOAD CASE, NAME=GlobalLoadCaseName2
*CLOAD
*END LOAD CASE
```

```
*LOAD CASE, NAME=GlobalLoadCaseName3
*CLOAD
*END LOAD CASE
*END STEP
```

The following usage example refers to two out of three linear load cases described in the previous global model usage example using the first method:

```
*STEP
*LOAD CASE, NAME=SubmodelLoadCaseName1
*SUBMODEL LINEAR LOAD CASE REFERENCE, GLOBAL LOAD
CASE=GlobalLoadCaseName1
*END LOAD CASE
*LOAD CASE, NAME=SubmodelLoadCaseName2
*SUBMODEL LINEAR LOAD CASE REFERENCE, GLOBAL LOAD CASE=
GlobalLoadCaseName3
*END LOAD CASE
*END STEP
```

The usage example below refers to the same two linear load cases of the global model as the previous submodel usage example. However, this example uses the second method, which increases simulation time compared.

```
*STEP
*SUBMODEL LINEAR LOAD CASE REFERENCE, GLOBAL LOAD CASE=
GlobalLoadCaseName1
*END STEP
*STEP
*SUBMODEL LINEAR LOAD CASE REFERENCE, GLOBAL LOAD CASE=
GlobalLoadCaseName3
*END STEP
```

Output

Submodel analysis is identical to any other analysis. You can request all output variables that are appropriate to the procedure.

Limitations

General submodeling is subject to the following limitations:

- Elements for forced convection/diffusion are not supported.
- Elements with coupled thermal-electrical fields are not supported.
- Piezoelectric elements are not supported.
- Electromagnetic elements are not supported.
- Coupled thermal-electrochemical, coupled thermal-electrochemical-structural, and coupled thermal-electrochemical-structural-pore pressure elements are not supported.
- Gasket elements, cylindrical elements, and surface elements are not supported.
- Special-purpose elements such as cohesive elements, fluid pipe elements, drag chain elements, Eulerian elements, pipe-soil elements, spring elements, and acoustic-interface elements are not supported.
- You cannot restart a general submodel analysis.

Input File Template

Submodel analysis definition:

```
*SURFACE
*SUBMODEL CUT, LOCAL SURFACE=element_based_surface
*SUBMODEL CUT, LOCAL SURFACE=element_based_surface

*STEP
*STATIC
...
*SUBMODEL TIME SYNCHRONIZATION
*SUBMODEL CONDITIONS
*LOAD CASE
*SUBMODEL LINEAR LOAD CASE REFERENCE
*END LOAD CASE
...
*END STEP
```

Node-Based Submodeling

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Submodeling*
- *Output to the Output Database*
- **SUBMODEL*
- **BOUNDARY*
- *Submodeling*

Overview

The following types of node-based submodeling that use the submodel interface are available:

- Same-to-same (for example, solid-to-solid, shell-to-shell);
- Shell-to-solid; and
- Acoustic-to-structure.

These submodel types support the following nodal-driven variables:

- Displacement,
- Rotation,
- Temperature,
- Pore pressure, and
- Acoustic pressure.

Another technique for node-based submodeling uses the field import interface.

Node-Based Submodeling

Two techniques are available for node-based submodeling: one technique uses the submodel interface and the other technique uses field import.

Node-Based Submodeling Using the Submodel Interface

Node-based submodeling using the submodel interface is a more general technique than surface-based submodeling. This technique supports a variety of element type combinations and procedures in both Abaqus/Explicit and Abaqus/Standard.

Input File Usage: **SUBMODEL*, TYPE=NODE

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Element Types Supported

You can use different element types in the submodel than those used to model the corresponding region in the global model.

The following types of submodeling are provided for the node-based approach (global-to-submodel):

- Two-dimensional models:
  - Solid-to-solid
  - Acoustic-to-structure
- Three-dimensional models:
  - Solid-to-solid
  - Shell-to-shell
  - Membrane-to-membrane
  - Shell-to-solid
  - Acoustic-to-structure

A global or submodel region that is meshed with continuum shell elements constitutes a three-dimensional solid region in the submodeling technique. Hence, the use of the submodeling technique for models involving continuum shell elements is the same as with models involving continuum solid elements such as C3D8R or C3D6.

Procedures Supported

Both the global model and the submodel can have nonlinear response and can be analyzed for any sequence of analysis procedures. These procedures do not have to be the same for both models. For example, the linear or nonlinear dynamic response of the global model can be used to drive the static, nonlinear response of the submodel on the grounds that the submodel is too small for dynamic effects to be significant in that local region. The global procedure can be performed in Abaqus/Standard to drive a submodeling procedure in Abaqus/Explicit and vice versa. For example, a static analysis performed in Abaqus/Standard can drive a quasi-static Abaqus/Explicit analysis in the submodel. The step time used in these analyses can be different; the time variable of the amplitude functions generated at the driven nodes can be scaled to the step time used in the submodel.

Your submodel cannot refer to a global model step that includes multiple load cases (see *Multiple Linear Load Case Analysis*). You must perform the global analysis with a single load definition for the step that will drive the submodel.

Node-Based Submodeling Using the Field Import Interface

Node-based submodeling using field import is an alternative approach for submodeling. Instead of using the submodel interface as described above, the field import interface is used to provide the necessary information. For each part of information that you provide using the submodel interface, an alternate way to provide the same part of information exists in the field import interface. For example, the global model name is provided as a parameter, the global element set is the source element set, and separate boundary conditions are not required. Node-based submodeling using field import allows for a large variety in the choice of driven variables in both Abaqus/Explicit and Abaqus/Standard. This approach reduces the preprocessor memory requirements, which is beneficial in very large models.

Input File Usage: **FIELD IMPORT*, SUBMODEL
**EXTERNAL FIELD*

Abaqus/CAE Usage: Node-based submodeling using the field import interface is not supported in Abaqus/CAE.

Element Types Supported

You can use different element types than those used in the submodel to model the corresponding region in the global model.

You can use the following types of submodeling for three-dimensional models with the node-based approach (global-to-submodel):

- Solid-to-solid
- Shell-to-shell
- Membrane-to-membrane

Procedures Supported

The node-based submodeling using field import method supports only time-based analysis procedures (see *General Capability for Importing External Fields*). You cannot use this method in a perturbation step.

Performing a Node-Based Submodeling Analysis

For an overview of submodeling that includes some details common to both node-based and surface-based submodeling, see *About Submodeling*.

Your submodel analysis is driven, either partly or completely, from the results obtained from a global model analysis. The results from the global model are interpolated onto the nodes on the appropriate parts of the boundary of the submodel (see *Figure 1*). Thus, the response at the boundary of the local region is defined by the solution for the global model. The driven nodes and any loads applied to the local region determine the solution in the submodel.

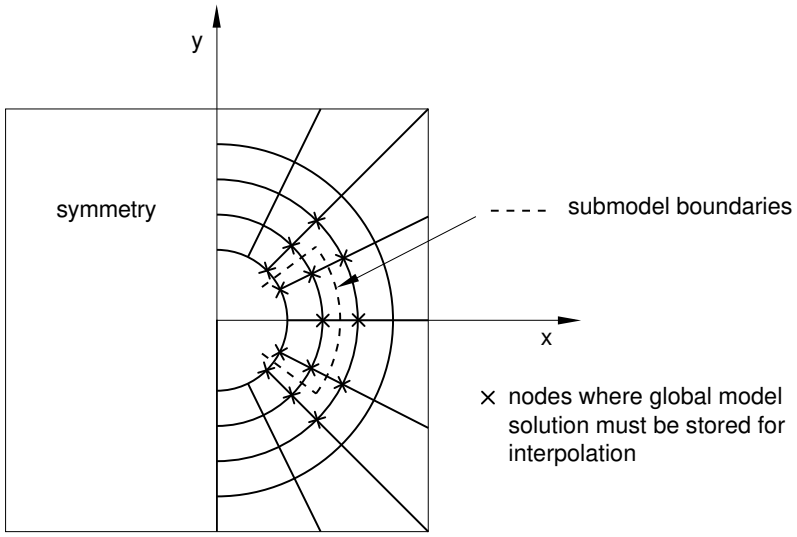


Figure 1: The global model.

Different Types of Node-Based Submodeling

Three different techniques are available for nodal-based submodeling.

Same-to-Same Submodeling

The linear or nonlinear response of a global model consisting of regions of solid, shell, or membrane elements can be used to drive the submodel response of similarly meshed regions of the submodel. The driven variables can be displacements or temperatures.

Shell-to-Solid Submodeling

The linear or nonlinear response of a global shell model can be used to drive the submodel response of a solid submodel. The driven variables are displacements, which are determined from global model displacements and rotations.

Acoustic Submodeling

The linear or nonlinear response of a global, structural model can be used to drive the acoustic response of a fluid region of any size if the forces exerted on the structure by the fluid are small. This is often the case for metal structures in air, building interiors, or for sound propagation from a liquid to air. In the case of a liquid and a gas, no special procedures need be followed; the pressure degrees of freedom couple straightforwardly. In the case of a structure driving a fluid, you must ensure that the degrees of freedom to be driven in the submodel exist among the global model results. Several alternatives exist. A thin layer of fluid elements, with the same properties as the submodel fluid, can be added to the global model; this element set and its nodes can then be used to drive the submodel in the usual manner. Alternatively, you can create acoustic interface elements on the surface of the submodel and drive the corresponding nodes with the structural nodes (see *Fully and sequentially coupled acoustic-structural analysis of a muffler*).

In problems where the fluid exerts large pressures on the structure, the mechanical response of the structure might be of interest. Acoustic-to-structure submodeling can be used in such problems. The submodel in these problems is a part of the structural component of the global model. The acoustic pressure obtained from solving a coupled acoustic-structural global analysis is used to drive the submodel on the surface it shares with the fluid medium. Other boundaries of the submodel might be driven using the displacements of the structural component of the global model via solid-to-solid submodeling. The acoustic-to-structure submodel analysis solves an uncoupled structural force-displacement problem. The acoustic pressure from the global model is interpolated to the submodel driven nodes. The tributary area and the outward normal associated with the driven node are used to convert the interpolated acoustic pressure to a concentrated load acting at that location (see *Miscellaneous submodeling tests*).

Specifying the Global Elements Used to Drive the Submodel

By default, the global model in the vicinity of the submodel is searched for elements that encompass the locations of driven nodes or driven surfaces' facets; the submodel is then driven by the response of these elements. In some cases more than one element can encompass the location of a driven node. For example, adjacent bodies in the global model might have temporarily coincident nodes or surfaces, as depicted in *Figure 2*.

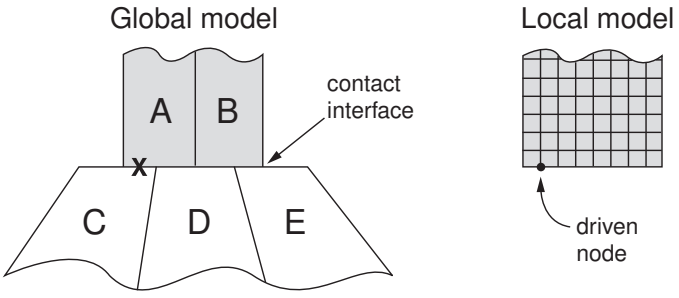


Figure 2: A global model with coincident surfaces in the area of the local model's driven nodes.

In this case the location of the driven node in the corresponding global model is touching both element A and element C; however, only the results from element A should drive the node in the submodel.

To preclude certain elements from driving the submodel, you can specify a global element set to limit the search to an appropriate subset of the global model.

Using the Submodel Interface

You provide the global element set to drive the submodel when using the submodel interface.

Input File Usage: **SUBMODEL*, GLOBAL ELSET=*name*

If the global model is defined in terms of an assembly of part instances, give the complete name—including the assembly and part instance names—when specifying the global element set. For example, an element set named `top` in part instance `I-1` of assembly `Assembly-1` must be referred to by `Assembly-1.I-1.top`.

If the submodel is not defined in terms of an assembly of part instances, the dots in the global element set name must be replaced by underscores: `Assembly-1_I-1_top`.

If the global element set is defined at the assembly level, you can provide the element set name without qualifying it with the assembly name in a submodel analysis.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Using the Field Import Interface

You provide the global element set as the source element set for the external field when using field import to drive the submodel analysis (available only for node-based submodeling).

Input File Usage: **FIELD IMPORT*, SUBMODEL
**EXTERNAL FIELD*
, , , , global_elset_name, ,

Abaqus/CAE Usage: Node-based submodeling using the field import interface is not supported in Abaqus/CAE.

Saving the Results from the Global Model

The results from the global analysis must be saved at all nodes required for the interpolation of the driven variables to the boundary of the submodel (see *Figure 1*). The results (`.fil`) file or the output database (`.odb`) file can be used for this purpose.

Saving the Results to the Results File

In each step of the global model whose solution will be used to drive the submodel, write the nodal results for all driven variables to the results file (see *Output to the Data and Results Files*). These results must be written in the global coordinate system of the model. The submodel can refer only to a global model results file that is from a binary compatible platform.

When the global model is run in Abaqus/Explicit and results file output is requested, the results are written to the selected results (`.sel`) file; this file needs to be converted into a results (`.fil`) file using the **convert** option (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Input File Usage: **NODE FILE*
(In Abaqus/Standard GLOBAL=NO should not be used on the **NODE FILE* option.)

Abaqus/CAE Usage: You cannot write output to the results file in Abaqus/CAE.

Saving the Results to the Output Database

In each step of the global model whose solution will be used to drive the submodel, write the nodal results for all driven variables to the output database in ODB or SIM format (see *Output to the Output Database*). Unlike the results file, nodal output to the output database is always written in the global directions. The output database can be transferred to any platform since it is binary neutral.

Input File Usage: Use both of the following options:

**OUTPUT*, FIELD
**NODE OUTPUT*

Abaqus/CAE Usage: Step module: **Output->Field Output Requests->Create**

Saving Results with Higher Precision

By default, the nodal output to the output database is written using single precision, which might not be sufficient for certain classes of problems; for example, submodels undergoing large rigid body motions (consider also surface-based submodeling in these cases—see *Surface-Based Submodeling*). For such analyses request the nodal output to the fullest possible precision (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Input File Usage: **abaqusjob**=*global_model_input_file* **output_precision**=`full`

Abaqus/CAE Usage: Job module: **Create Job**: **Precision**: **Nodal output precision**: **Full**

Saving Results from a Global Model with a Physical Time Scale

If the global analysis in Abaqus/Standard involves a physical time scale and the results file is to be used in the submodel analysis, request that the results file output be written at the beginning of the step (the zero increment) for all steps in the global analysis (see *About Output*). Abaqus will then have the complete solution history (including the solution state at the beginning of a step) from which a submodel might be driven. If the zero increment results are not requested, incorrect results will be obtained if the step time in the submodel is less than the step time of the first increment on the results file. Instead of interpolating between the results at the start of the step and the results of the first increment on the results file, Abaqus will simply use the results of the first increment as long as the submodel step time is less than the step time of the first increment on the results file. The zero increment request is not required in Abaqus/Explicit, because the results are always written to the results file at the beginning of each step. Similarly, the results will always be correctly interpolated when using the output database to transfer the results from the global model to the submodel, because the zero increment is always written to the output database.

Input File Usage: **FILE FORMAT*, ZERO INCREMENT

Abaqus/CAE Usage: You cannot write output to the results file in Abaqus/CAE.

Referring to the Global Model Results from the Submodel Analysis

You must define the source of the global solution results.

Using the Submodel Interface

You provide the name of the global results file or output database file (in ODB or SIM format); the file extension is optional. If the file extension is omitted, Abaqus uses in order, the results file, the ODB output database file, or the SIM database file.

Input File Usage: **abaqusjob=***submodel_input_file* **globalmodel=***global_results_file* or *global_output_database* or *sim_database_file*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*submodel*: Submodel**: **Read data from job**: *global_results_file* or *global_output_database*

Using the Field Import Interface

You provide the SIM database name to refer to the global model results when using the field import interface.

Input File Usage: **EXTERNAL FIELD*, FILE=*file_name*`.sim`

Abaqus/CAE Usage: Reading data from a SIM database file is not supported in Abaqus/CAE.

Specifying the Driven Nodes in the Submodel

You specify the driven nodes in the submodel.

Using the Submodel Interface

Specifying the driven nodes does not activate the driven variables: they must be activated by specifying the appropriate submodel boundary conditions.

All nodes of the submodel where variables will be driven in any step (see *Figure 3*) must be specified as driven nodes since the list of nodes cannot be extended subsequent to its initial definition (even at restart). However, variables at the nodes given do not have to be driven in all steps: the choice of which variables are driven in a particular step is made as part of a submodel boundary condition definition, as discussed later.

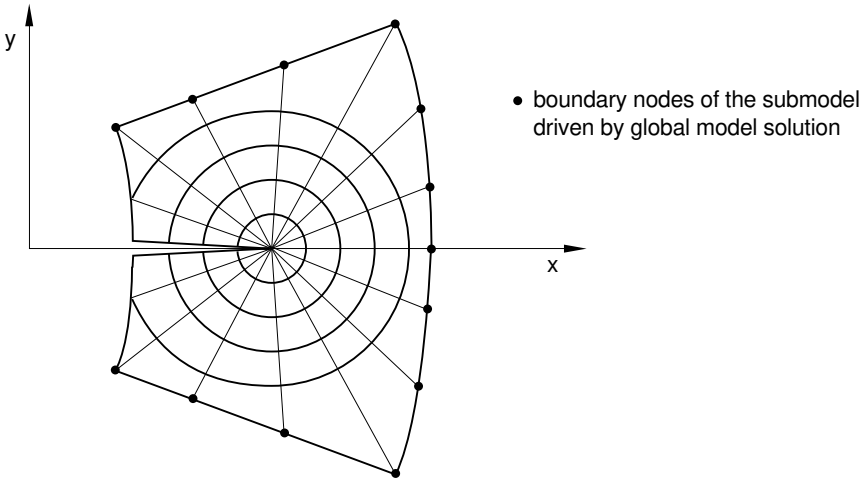


Figure 3: The magnified submodel.

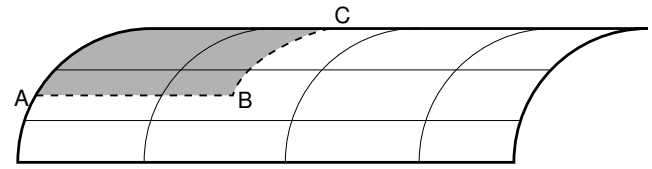
Input File Usage: **SUBMODEL*
list of nodes or node set labels or, for acoustic-to-structure submodeling, the name of an element-based structural surface

The **SUBMODEL* option must be included in the model definition portion of the input file for the submodel analysis. Multiple **SUBMODEL* options are allowed; however, in this case you must ensure that the driven nodes specified on the data line of one option are separate and distinct from the nodes specified on the data lines of all the other options.

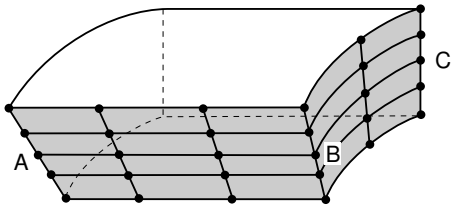
Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region

Specifying the Driven Nodes in Shell-to-Solid Submodeling

In shell-to-solid submodeling, the submodel is made up of solid elements and replaces a region where conventional shell elements are used in the global model. In this case Abaqus expects that all the driven nodes on the submodel belong to solid elements and are driven from a global model region that is entirely made up of shell elements. The boundary where the submodel is driven is a set of surfaces in the submodel but is a set of lines in the shell reference surface in the global model, as shown in *Figure 4*. The dashed line $A - B - C$ on the shell model is replaced by the shaded surfaces of the solid element submodel.



a) Shell global model with submodel boundaries



A, B, C - shell reference surface
● - driven nodes

b) Magnified solid element submodel

Figure 4: Shell-to-solid submodeling.

Whenever shell-to-solid submodeling is used, you must define the maximum shell thickness in the global model, given in the units used for the models. If a shell offset is defined in the global model, the shell thickness must be set equal to twice the maximum distance from the top or bottom shell surface to the shell reference surface.

Input File Usage: **SUBMODEL*, SHELL TO SOLID, SHELL THICKNESS=*thickness*

If more than one **SUBMODEL* option is used, the SHELL TO SOLID parameter must be included on every option.

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*submodel***: **Submodel**: **Shell global model drives a solid submodel**
Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Shell thickness:** *thickness*

Specifying the Driven Nodes in Acoustic-to-Structure Submodeling

The global analysis for acoustic-to-structure submodeling problems is performed as a coupled acoustic-structural analysis. The acoustic nodal pressures from the global analysis must be written to the results file for the acoustic

mesh in contact with the structural surface of interest. In the submodel analysis acoustic pressures from the global analysis drive the user-specified structural surface of interest. The driven nodes for the submodel are the nodes lying on the specified surface. Only element-based surfaces are allowed in acoustic-to-structure submodeling.

Input File Usage: **SUBMODEL*, ACOUSTIC TO STRUCTURE,
ABSOLUTE EXTERIOR TOLERANCE=*value*

Abaqus/CAE Usage: Acoustic-to-structure submodeling is not supported in Abaqus/CAE.

Specifying Driven Nodes for Shells with Acoustic Pressures on Both Sides

In certain problems the acoustic pressure might act on both sides of a shell structure. *Figure 5* shows a section of a global model consisting of a shell structure that is sandwiched between two acoustic media.

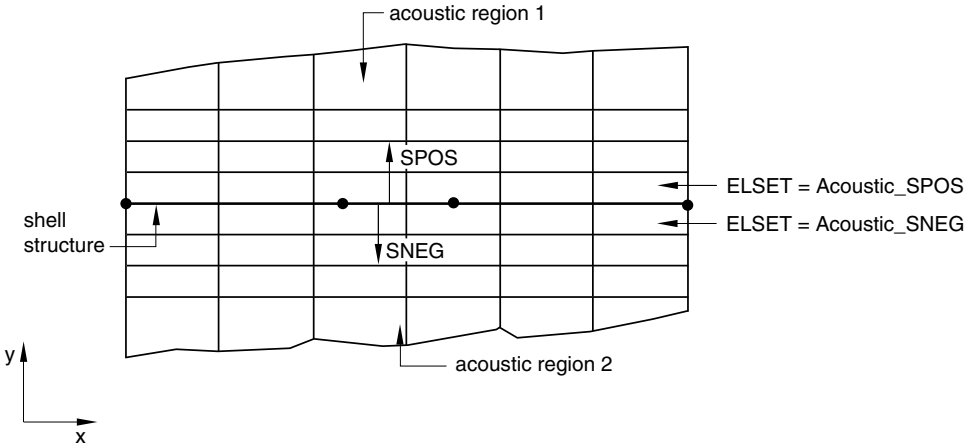


Figure 5: A cross-section of the acoustic-to-structure global model with acoustic regions on both sides of the shell.

Separate element sets consisting of acoustic elements on the positive and negative sides of the shell are defined, respectively. The nodal pressures for nodes attached to elements in these sets are written to the selected results file. *Figure 6* shows the submodel that consists only of the refined shell structure.

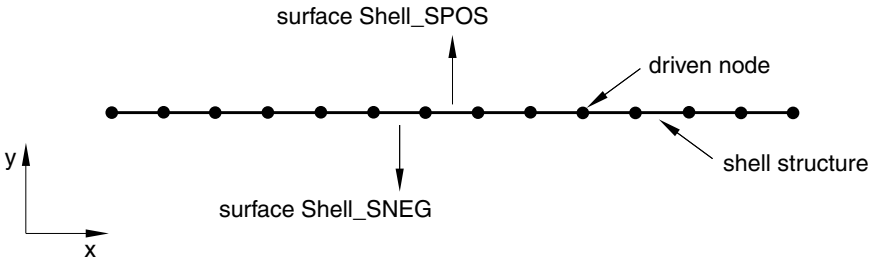


Figure 6: The acoustic-to-structure submodel with acoustic pressure on both sides of the shell.

Two separate surfaces are defined on the SPOS and SNEG sides, respectively. To apply the acoustic pressure from the global analysis on each side of the shell correctly, you must specify the surface name along with the corresponding acoustic element set.

Input File Usage: **SUBMODEL*, ACOUSTIC TO STRUCTURE, GLOBAL ELSET=*Acoustic_SPOS Shell_SPOS*

**SUBMODEL*, ACOUSTIC TO STRUCTURE, GLOBAL ELSET=*Acoustic_SNEG Shell_SNEG*

Abaqus/CAE Usage: Acoustic-to-structure submodeling is not supported in Abaqus/CAE.

Defining Geometric Tolerances

A geometric tolerance is used to define how far a boundary node in the submodel can lie outside the exterior surface of the global model, as that surface is interpolated in the global, undeformed finite element model. By default, nodes in the submodel must lie within a distance calculated by multiplying the average element size in the global model by 0.05. You can change the tolerance, which is useful in cases where submodel driven nodes lie to a greater extent outside the global model exterior surface. Tolerances larger than this default value, however, might result in significantly greater computation times and lower accuracy in the driven solution for driven nodes significantly outside the global model exterior surface.

You can define the geometric tolerance as a fraction of the size of the average element in the global model or as an absolute distance in the length units chosen for the model. If both tolerances are defined, Abaqus uses the tighter tolerance.

Input File Usage: Use the following option to define the geometric tolerance as an absolute distance:

**SUBMODEL*, ABSOLUTE EXTERIOR TOLERANCE=*tolerance*

Use the following option to define the geometric tolerance as a fraction of the size of the average element in the global model:

**SUBMODEL*, EXTERIOR TOLERANCE=*tolerance*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Exterior tolerance: absolute:** or **relative:** *tolerance*

The Exterior Tolerance in Solid-to-Solid Submodeling

The exterior tolerance for a solid-to-solid submodel analysis is indicated by the shaded region in *Figure 7*. If the distance between the driven nodes and the free surface of the global model falls within the specified tolerance, the solution variables from the global model are extrapolated to the submodel.

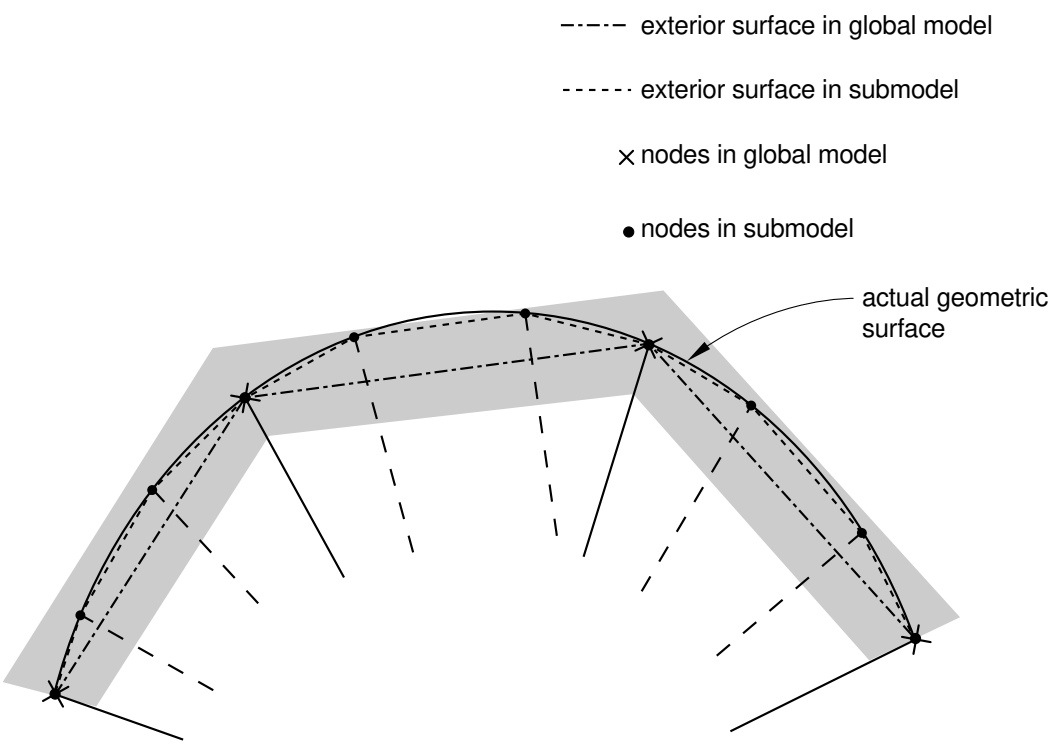


Figure 7: The exterior tolerance in solid-to-solid submodeling.

The Exterior Tolerance in Shell-to-Shell Submodeling

In a shell-to-shell submodel analysis Abaqus checks whether the driven nodes of the submodel lie sufficiently close to the reference surface of the shell elements in the global model. To simplify calculations, the closest point in the global model is calculated as the intersection of a line drawn through the node on the submodel with the reference surface of the shell in the global model. The direction of the line is normal to a flat surface approximation to each shell element. The normal to the flat surface is the average of the normals at the nodes of the shell element. The distance checked against the specified exterior tolerance is shown in *Figure 8*.

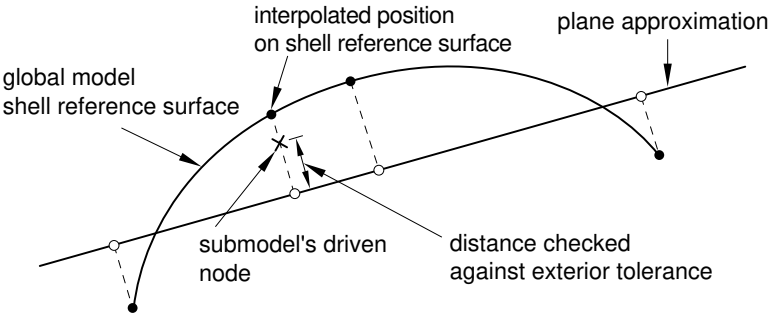


Figure 8: Flat surface approximation in shell-to-shell submodeling.

The Exterior Tolerance in Shell-to-Solid Submodeling

For shell-to-solid submodeling Abaqus uses two kinds of tolerances to determine the relationship between the submodel and the global model. First, the closest point on the shell reference surface of the global model is determined. This point, the “image node,” is shown in *Figure 9*. The user-specified exterior tolerance is used to check if the image node lies within the domain of the global model. Then the distance, $\boldsymbol{D}$, between the driven node and its image is checked; if the distance is less than half the value of the specified shell thickness plus the

exterior tolerance, it is accepted. This check is only approximate if the global model has varying shell thickness or if the shell reference surface is offset from the midsurface.

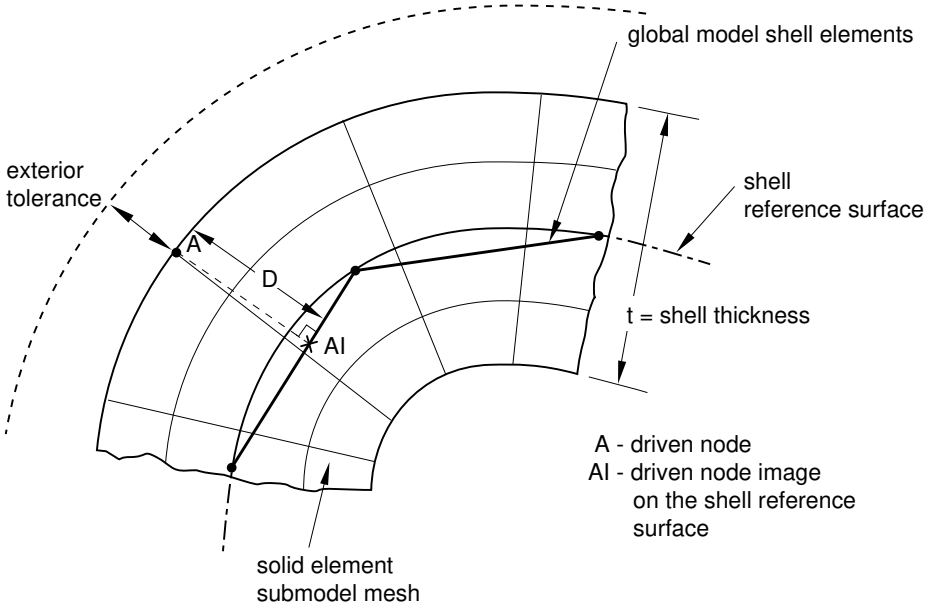


Figure 9: The exterior tolerance in shell-to-solid submodeling.

Permitting Driven Nodes to Be Excluded from Submodeling

In some cases (such as when your submodel geometry is more detailed than the global model in regions near a free surface) you may specify driven nodes that Abaqus will find, even when accounting for the search tolerance, to be outside the region of the global model elements. By default, these cases result in an error message. In solid-to-solid submodeling you can, however, specify that Abaqus ignore driven nodes that cannot be found. Use this option with caution and always evaluate the list of nodes that are labeled as not found. Most cases where Abaqus finds driven nodes to lie outside the global model reflect a modeling error and use of the intersection only option might lead to incorrect results in these cases.

Input File Usage: Use the following option to specify that Abaqus ignore driven nodes that cannot be found in the global model elements:

**SUBMODEL*, INTERSECTION ONLY
list of nodes or node set labels

The driven nodes ignored through the use of the INTERSECTION ONLY parameter are then ignored in all subsequent submodel boundary condition references.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Intersection only**

Using the Field Import Interface

You specify the driven node set as the target node set for the external field when using the field import interface.

Input File Usage: **FIELD IMPORT*, SUBMODEL
**EXTERNAL FIELD*
, *target node set name*,

Abaqus/CAE Usage: Node-based submodeling using the field import interface is not supported in Abaqus/CAE.

Defining the Driven Variables in the Submodel

You define the driven variables in the submodel.

Using the Submodel Interface

The actual driven variables are defined in any step as a submodel boundary condition. The boundary conditions are “driven variables” obtained from the results or output database file of the global analysis.

The degrees of freedom on the driven nodes of the submodel must exist at the forcing nodes of the global model. In a problem involving an acoustic fluid submodel driven by a structural global model, for example, acoustic interface elements should be created on the submodel's driven boundary with the structure.

For solid-to-solid and shell-to-shell submodeling specify the individual degrees of freedom to be driven. In most cases all components of the solution variables (displacements, rotations, temperatures, etc.) at these nodes are driven by the global solution, although you might choose to drive only some components at any of the driven nodes. For shell-to-solid submodeling the driven degrees of freedom are chosen automatically based on a user-specified zone around the shell reference surface, as explained later.

Abaqus/Explicit does not admit jumps in displacement and rotation boundary conditions (see *Prescribed Displacement*); any jumps in the driven displacements and rotations will be ignored.

It is not recommended to have all the variables at all the nodes in the submodel driven by the global solution.

For acoustic-to-structure submodeling, the loads due to acoustic pressure acting at the driven nodes of the submodel are activated by specifying pressure (degree of freedom 8) along with the driven node set.

Only one submodel boundary condition can be specified in each step of the analysis.

Input File Usage: **BOUNDARY*, SUBMODEL

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Degrees of freedom:** *degrees of freedom*

Specifying the Step Number from the Global Analysis

You specify the step of the global model history that is to be used for the driven variables in the current submodel analysis step. When the global solution is obtained from the results file, the zero increment is included if it was requested in the global analysis (see *About Output*).

In a general analysis step or a direct-solution steady-state dynamic analysis step, Abaqus calculates the amplitudes for the driven variables as functions of time or frequency from the results of the global model.

Input File Usage: **BOUNDARY*, SUBMODEL, STEP=*step*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number:** *step*

Scaling the Global Time Period to the Submodel Time Period

The global analysis and submodel analysis can have different time steps. You can scale the time variable of the driven nodes from the global analysis to the step time of the submodel analysis. This technique is useful when the analyses are static or quasi-static in nature; the use of this technique in dynamic analyses with significant inertial effects is not recommended. If the same step time is used in both the global model and the submodel, the time scale has no effect. The time scale cannot be specified in frequency domain analyses or in linear perturbation steps.

Abaqus will determine the values that the driven variables will follow throughout the step in the submodel analysis by using the points in time at which the global solution results or output database file was written. When the time variable of the driven nodes of the global analysis is scaled and if the step time is different from the submodel step time, the points in time of the driven variables are scaled to the submodel step time.

Input File Usage: **BOUNDARY*, SUBMODEL, STEP=*step*, TIMESCALE

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Scale time period of global step to time period of submodel step**

Scaling the Magnitude of Driven Variables

For displacement-based submodeling the magnitude values of driven variables are obtained by multiplying the displacement history as obtained from the global analysis by a scaling parameter. You can scale the driven variables by setting the scaling parameter in the definition of the submodel boundary conditions. This technique is useful in scaling the submodel boundary conditions in a multiple-step analysis without rerunning the global model. It can be used in Abaqus/Standard and Abaqus/Explicit for the same-to-same and shell-to-solid cases except for acoustic-to-structure submodeling.

Input File Usage: **BOUNDARY*, SUBMODEL, STEP=*step*, SCALE=*scalarValue*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Scale:** *scale*

Modifying the Set of Driven Variables

You can modify the submodel boundary condition to add new variables to the list of driven variables, you can remove variables from the driven variable set, and you can reintroduce them later (see *Boundary Conditions*). If a submodel boundary condition is not redefined in a new step, the driven variables remain constant at the magnitude associated with the end of the previous step. New nodes cannot be added to the total set of driven nodes defined for the submodel; this set of driven nodes is a fixed part of the model definition.

Input File Usage: Use one of the following options:

**BOUNDARY*, SUBMODEL, OP=MOD
**BOUNDARY*, SUBMODEL, OP=NEW

Abaqus/CAE Usage: Load module: boundary condition editor: **Degrees of freedom**

Automatically Selecting the Driven Variables in Shell-to-Solid Submodeling

For shell-to-solid submodeling the driven degrees of freedom at the driven nodes are chosen automatically, depending on the distance between the driven node and the global model shell reference surface. All displacement components are driven at nodes that lie on the reference surface or within a “center zone,” as shown in *Figure*

10. The size of the center zone is specified as part of the submodel boundary condition definition, as described below. For nodes that lie further away from the reference surface, only the displacement components parallel to the shell reference surface are driven. At least one layer of nodes in the submodel must be within the center zone; if no nodes are found this close to the reference surface, Abaqus issues an error message.

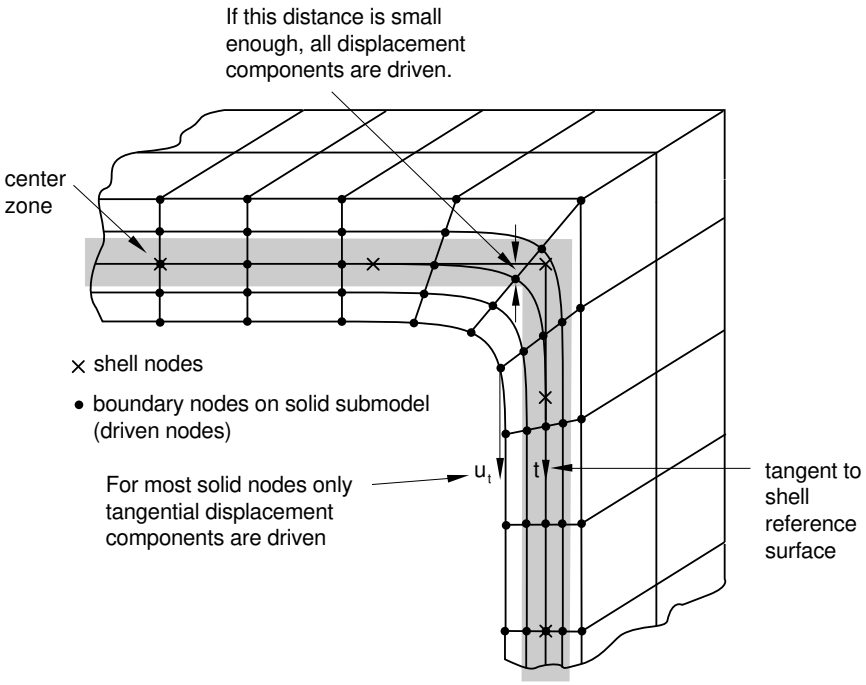


Figure 10: Center zone choice in shell-to-solid submodeling.

Specifying the Size of the Center Zone in Shell-to-Solid Submodeling

The center zone method of prescribing driven variables usually provides a reasonable transfer of the plane stress assumption in the shell model. The width of this zone around the reference surface where all displacement components are driven might be different for various driven nodes or node sets. If you do not provide values for the center zone size, a default value of 10% of the maximum of the specified shell thicknesses is assumed.

For complicated geometries it can be advantageous to assign a different center zone size to different nodes or node sets.

You can view the driven nodes lying inside and outside the center zone in Abaqus/CAE by displaying the model boundary conditions (**View->ODB Display Options**) in the Visualization module.

Input File Usage: **BOUNDARY*, SUBMODEL, STEP=*step*
nodes, *center zone size*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Center zone size:** *center zone size*

Transferring Transverse Shear Stresses in Shell-to-Solid Submodeling

Usually it is enough for the layer of nodes closest to the shell reference surface to lie inside the center zone. If a very fine solid mesh is used in the thickness direction and substantial transverse shear stresses are transferred, it might be necessary to make the center zone size large enough that multiple layers of nodes lie inside the zone.

However, if the transverse shear stresses at the submodel boundary are high and the submodel is highly refined in the thickness direction, high local stresses might develop since the shear force at the submodel boundary is transferred only at the driven nodes within the center zone. High transverse shear stresses occur only in regions where bending moments vary rapidly; it is better not to locate the submodel boundary in such regions. It is best to locate the submodel boundary in areas of low transverse shear stress in the global model.

Using the Field Import Interface

You specify the driven variables as part of the data defining the external field for submodeling using the field import interface. The step time from the global model is scaled to match the time period of the submodel.

Input File Usage: **FIELD IMPORT*, SUBMODEL
**EXTERNAL FIELD*
, , *driven variable*,

Abaqus/CAE Usage: Node-based submodeling using the field import interface is not supported in Abaqus/CAE.

Special Considerations

There are several special considerations that are worth noting.

Limitations Using the Node-Based Submodeling Field Import Interface

The node-based submodeling field import interface has the following limitations:

- You cannot include a step following a step in which a submodel is defined.
- You cannot use a submodel job to restart a subsequent job or to import to a subsequent job.
- Submodeling is not supported in linear perturbation procedures, frequency-based procedures, or a geostatic procedure.
- A submodel job cannot contain a mixture of two-dimensional and three-dimensional elements.
- Two-dimensional, axisymmetric, and generalized plane-strain elements are not supported.
- Submodeling in a frequency domain is not supported.
- Temperature degrees of freedom cannot be driven for shell elements.
- Acoustic-to-structure submodeling is not supported.

Specifying the Shell Thickness in Shell-to-Shell Submodeling

For shell-to-shell submodeling the shell thickness generally is not changed between the models. You can specify different shell thicknesses if, for example, a local thickness change is being investigated; however, Abaqus does not check the validity of these differences.

Limitations in Shell-to-Solid Submodeling

The following limitations and special cases apply to the shell-to-solid capability:

- The global model can contain both solid and shell elements; however, when the shell-to-solid capability is used, all driven nodes must lie within shell elements in the global model. If the driven boundary lies at the border between a solid and a shell region, the driven nodes must be moved a small distance away from the solid region (see *Figure 11*).

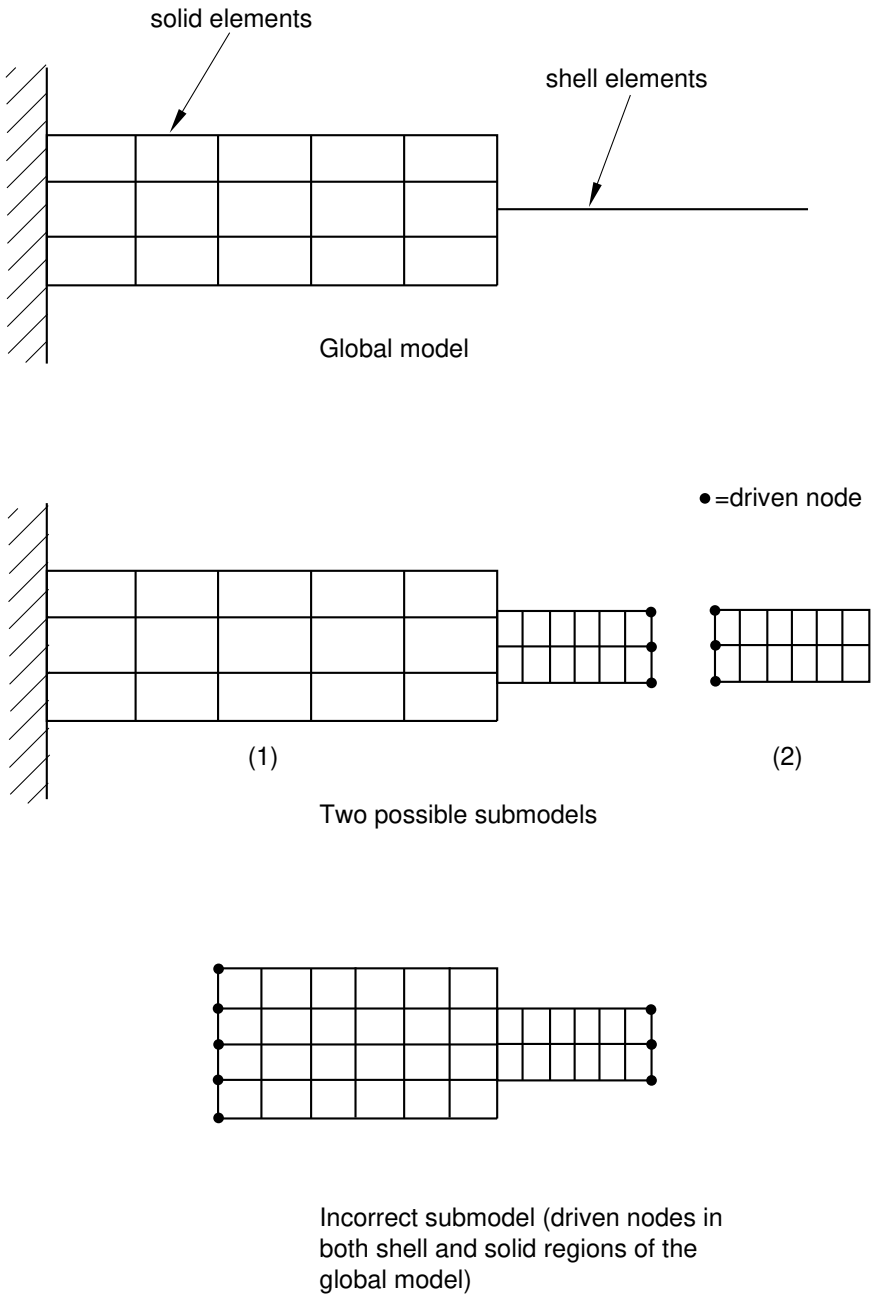


Figure 11: A limitation of shell-to-solid submodeling.

- Corners or kinks might exist in global models made of shell elements. At such corners or kinks the shell elements only approximate the distribution of the material away from the midsurface of the shell (see *Figure 12*). Because of such approximations, it is not possible to drive a submodel correctly if the driven nodes of the submodel lie within a shell thickness from a corner or a kink. If necessary, use the approach shown in *Figure 12*.

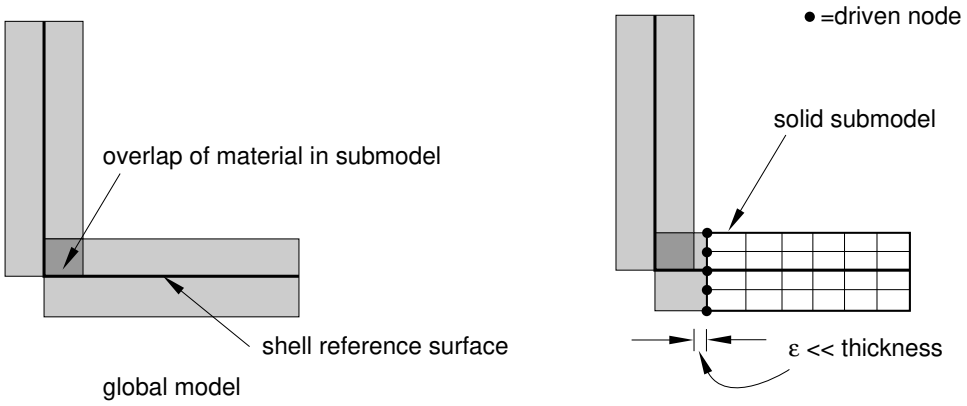


Figure 12: Shell-to-solid submodeling around corners.

A better approach is to include the corner or kink as part of the submodel and drive it from nodes well away from corners or kinks since they are a source of stress concentration and high stress gradients (see *Figure 13*).

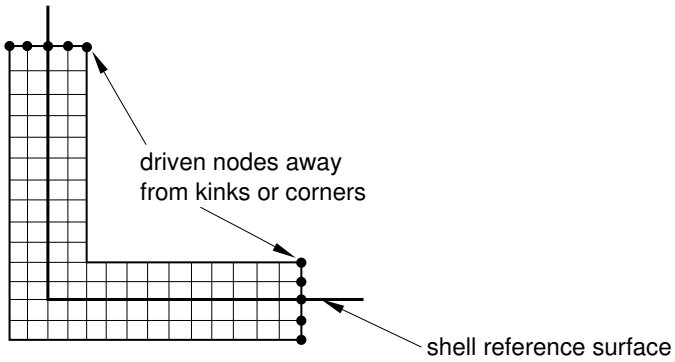


Figure 13: Solid submodel of a shell intersection.

- Temperature degrees of freedom cannot be driven in shell-to-solid submodeling.

Alternative to Shell-to-Solid Submodeling

An alternative to shell-to-solid submodeling is the surface-based shell-to-solid coupling capability discussed in *Shell-to-Solid Coupling*.

Procedures

Neither the coupled thermal-electrical procedure nor any of the mode-based dynamics procedures can be used on the submodel level. In addition, submodeling cannot be used in conjunction with symmetric model generation or symmetric results transfer. Adaptive meshes should not be used in the global model. However, they can be used in the submodel analysis; Abaqus will always treat the driven nodes in the submodel as Lagrangian nodes.

Both general (possibly nonlinear) and linear perturbation steps can be used in submodeling (see *General and Perturbation Procedures* for a discussion of general and linear perturbation steps).

Submodeling in Dynamic Procedures

The submodeling capability can be used in the dynamic procedures using explicit integration (in Abaqus/Explicit) and in the dynamic procedures using direct integration (in Abaqus/Standard). The following combinations of procedures between the global model and the submodel can be considered: explicit dynamic, implicit dynamic,

dynamic coupled thermal-stress, and coupled thermal-stress. In dynamic problems in which inertial forces are significant, the global model and the submodel need to be run for the same step time intervals.

In Abaqus/Explicit a quasi-static analysis is performed as a dynamic procedure. For this case and for the static analyses performed in Abaqus/Standard, the time step of the global model and submodel can be different. The time variable of the driven nodes from the global analysis must be scaled to the step time of the submodel analysis to match the time variable of the amplitude functions generated at the driven nodes to the step time used in the submodel.

For significantly dynamic problems in Abaqus/Explicit, a sufficiently large number of intervals need to be written to the results or output database file for the global model. Preferably the displacement results for the nodes that are used to drive the submodel should be saved for each increment. This caution is necessary in particular for problems with elastic material properties to avoid possible aliasing (under sampling), which can cause solution distortion in the submodel. These requirements do not apply to quasi-static problems.

Interpreting Acceleration Results

When you drive a submodel boundary with global model displacement results, the displacements are interpreted as a smoothed piecewise linear function in time, similar to how you would apply a displacement boundary condition using a tabular amplitude definition (see *Using an Amplitude Definition with Boundary Conditions*). This smoothed function typically results in displacements and velocities at the driven nodes that agree reasonably with the global model. Acceleration results at the driven boundary, however, are generally not in good agreement with the global model as they reflect the shape of the displacement history smoothing rather than the global model acceleration results (information that is not available from a piecewise linear global-model displacement history). The submodel acceleration results away from the submodel driven nodes are less affected by this smoothing and are typically in good agreement with the global model response.

Obtaining a Solution at a Particular Point in Time Using Linear Perturbation Analysis

In Abaqus/Standard it is possible to study the submodel's linearized response corresponding to a particular point in time in the global solution by using a static, linear perturbation procedure in the submodel analysis. You can select the increment in the global analysis step that is to be used as the basis for calculating the values for the driven variables. If you do not select an increment in a static linear perturbation step, the last increment of the selected step in the global analysis is used as the basis for calculating the values for the driven variables. You cannot select an increment in a general submodel step.

Input File Usage: **BOUNDARY*, SUBMODEL, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number**: *step*, **Global increment**: *increment*

Submodeling in the Frequency Domain

The submodeling capability can be used in the frequency domain. In this case the global model can be run using the mode-based or direct-solution steady-state dynamic procedure and the submodel can be run only using the direct-solution steady-state dynamic procedure. A frequency-domain submodel cannot be driven with a time-domain global model or vice versa. Mode-based steady-state dynamics cannot be used at the submodel level.

The only restriction on the specification of the frequency range in the submodel is that the minimum and maximum frequency should lie within the range of calculated frequencies in the global model. Abaqus interpolates the solution variables from the global model in the frequency domain, as well as spatially, before applying them to the submodel. The results will be most accurate if the frequencies at which the response in the submodel is

requested match the frequencies at which the response was calculated in the global model. This is particularly true in the vicinity of the eigenfrequencies of the global model.

In the global model you must write both the amplitude and the phase of the nodal displacements to the results file so that Abaqus can apply the real and imaginary parts of the solution at the driven nodes in the submodel. If you are using the output database to drive the submodel, you need to request only nodal displacement output since displacement output to the output database includes both real and imaginary parts.

Mixing General and Linear Perturbation Steps

It is possible to mix general steps and linear perturbation steps in both the global and the submodel analyses. Abaqus allows general analysis steps to be treated as linear perturbation steps during submodeling, and vice versa.

Example: Submodeling with General and Linear Perturbation Steps

For an example of submodeling that uses both general and linear perturbation steps, consider the following situation. The global analysis consists of a static preload—done as a general, nonlinear, analysis step—followed by extraction of the eigenmodes of the preloaded structure, then a step of 5 seconds of modal dynamic response analysis:

```
*STEP
** Apply preload
*STATIC
 0.1, 1.0
…
** Write out results for nodes needed to
** interpolate to the submodel's boundary
*NODE FILE, NSET=DETAIL
 U
*END STEP
*STEP
** Calculate modes and frequencies
*FREQUENCY
…
** The *NODE FILE option is repeated because
** this is the first linear perturbation step
*NODE FILE, NSET=DETAIL
 U
*END STEP
*STEP
** Dynamic response of preloaded system
*MODAL DYNAMIC
 0.01, 5.0
…
*END STEP
```

We wish to study the local, possibly nonlinear, response of a part of this model that is so small that we do not need to model dynamic effects locally and can, thus, perform two steps of static analysis:

```
** Define submodel boundary (driven nodes)
*SUBMODEL
PERIM
*STEP
** Preload
*STATIC
 0.1, 1.0
*BOUNDARY, SUBMODEL, STEP=1
…
*END STEP
```

```
*STEP
** Local static response to global dynamic step
*STATIC
 0.01, 5.0
*BOUNDARY, SUBMODEL, STEP=3
…
*END STEP
```

It is perfectly acceptable that the submodel analysis requests general, possibly nonlinear, analysis for both steps, while in the global analysis the dynamic step was a linear perturbation step (modal dynamics is always a linear perturbation analysis). It is your responsibility to judge that this use of the submodeling feature is reasonable. For example, suppose that the global analysis were continued with a fourth step of general, nonlinear static response:

```
*RESTART, READ, STEP=3
** Read state at end of initial preload
** (could equally well use *RESTART, READ, STEP=1)
*STEP
** Add more preload
*STATIC
 0.2, 1.0
…
*END STEP
```

This fourth general analysis step starts with the state at the end of general analysis Step 1 because the frequency extraction and the modal dynamic steps are both linear perturbation steps. However, if we restart the submodel analysis in the same way, the solution might not be comparable with the global model solution:

```
*RESTART, READ, STEP=2
** Read state at end of step 2
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*BOUNDARY, SUBMODEL, STEP=4
…
*END STEP
```

The second step in the submodel is a general analysis step, to which the response might be nonlinear, thus changing the state of the model. A valid alternative would be to apply the Step 4 response to the submodel immediately after the first step:

```
*RESTART, READ, STEP=1
** Read state at end of preload step
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*BOUNDARY, SUBMODEL, STEP=4
…
*END STEP
```

Reinterpreting Solution Variables in the Submodel Analysis

During general analysis steps Abaqus works in terms of total solution variables such as the displacements, $\boldsymbol{u}$. In linear perturbation steps Abaqus works in terms of the displacement perturbation, $\Delta\boldsymbol{u}$, about a base state, $\boldsymbol{u}|_0$. When general analysis steps and linear perturbation steps are reinterpreted in the submodel analysis, the global analysis results are treated as defined in *Table 1*.

Table 1: Reinterpreting solution variables in the submodel analysis.

Global analysis step basis	Submodel step basis	Global increment specified in definition of submodel boundary condition	Driven variable basis
General	General	none	$u_s = u_g$
Linear perturbation	General	none	$u_s = u_s\|_0 + \Delta u_g$
General	Static, linear perturbation	i	$\Delta u_s = u_g\|_i - u_s\|_0$
Linear perturbation	Static, linear perturbation	i	$\Delta u_s = \Delta u_g\|_i$

In this table

u_s

is the current value of a driven variable in the submodel at any time during a general, nonlinear, analysis step;

Δu_s

is the value of the perturbation of a driven variable in the submodel during a linear perturbation step;

u_g **and** Δu_g

are the corresponding values of the same (geometrically interpolated) variable in the global model;

$u_g|_0$

is the "base state" value of the variable during a linear perturbation step in the global analysis;

$u_s|_0$

is the "base state" value of the variable during a linear perturbation step in the submodel analysis;

$u_g|_i$

is the value of u_g at increment *i* of the global analysis step; and

$\Delta u_g|_i$

is the value of Δu_g at increment *i* of the global analysis step.

Mixing General and Linear Perturbation Steps in Shell-to-Solid Submodeling

Additional assumptions must be made for the shell-to-solid case when a general procedure on the global model drives a linear perturbation procedure on the submodel and vice versa. The assumptions depend on the geometric formulation used (linear or nonlinear) and on the procedure combination. For details and governing equations for these cases, see *Submodeling analysis*.

Initial Conditions

The definition of initial conditions should be consistent between the global model and the submodel.

Boundary Conditions

Boundary conditions (other than submodel boundary conditions) prescribed on the degrees of freedom that are driven will replace those prescribed using submodel boundary conditions. When this replacement occurs, Abaqus reports the change in the data file.

A node can be driven from the global model in some steps and have user-prescribed boundary conditions in other steps. In these cases all relevant boundary conditions must be respecified (see *Boundary Conditions*).

Any other boundary conditions that are applied in the submodel region should be imposed in the submodel analysis in the usual way. It is your responsibility to apply such prescribed boundary conditions to the submodel correctly so that they correspond to the loading of the global model.

Be careful with submodel boundary nodes that are also on planes of symmetry, where both forms of boundary conditions can be applied. It might be helpful in such cases to apply boundary conditions in a local coordinate system (see *Transformed Coordinate Systems*). The local coordinate system should be applied only to the boundary conditions that are intended to override the submodel boundary conditions, since the submodel boundary conditions are always output in the global coordinate directions by the global model.

Loads

Any loads that are applied in the submodel region must be imposed in the submodel analysis in the usual way. It is your responsibility to apply such loads to the submodel correctly so that they correspond to the loading of the global model. See *About Loads* for an overview of the loads available in Abaqus.

Predefined Fields

The following predefined fields can be specified in a submodeling analysis, as described in *Predefined Fields*:

- Nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material (*Thermal Expansion*). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Abaqus interpolates solution variables onto the submodel driven nodes. It can also interpolate temperatures as field variables (see *Interpolating Data between Meshes* for details). Other predefined fields will not be interpolated to the nodes of the submodel; they must be available from the input data for all nodes of the submodel where they are required.

Abaqus/Standard provides multiple approaches for cases where a submodel thermal-stress analysis must be performed using temperature solutions from a global heat transfer analysis.

- Run a heat transfer analysis of the global model, and write the nodal temperatures to the results or output database file. Run a sequentially coupled thermal-stress analysis of the global model. The temperatures obtained from the results or output database file of the global heat transfer analysis are field variables in this case. If the mesh used in the thermal-stress analysis is different from the mesh in the heat transfer analysis, specify that Abaqus/Standard should interpolate the temperature field from the heat transfer analysis mesh to the thermal-stress analysis mesh. Run a thermal-stress analysis of the submodel using the results or output database file for the global thermal-stress analysis to read the driven variables (displacement field) and using the results or output database file from either the global heat transfer analysis or the global thermal-stress analysis to read the temperatures as field variables. In either case the temperature field will have to be

interpolated to the current submodel nodes. If interpolation between dissimilar meshes is necessary, the global output database file must be used to read the temperatures. For details, see *Figure 14* and *Figure 15*.

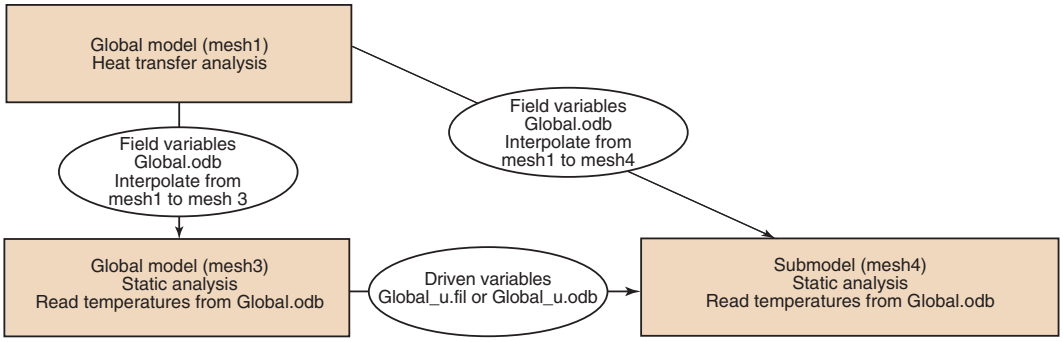


Figure 14: Sequentially coupled thermal-stress analysis for the global model with only a thermal-stress analysis for the submodel.

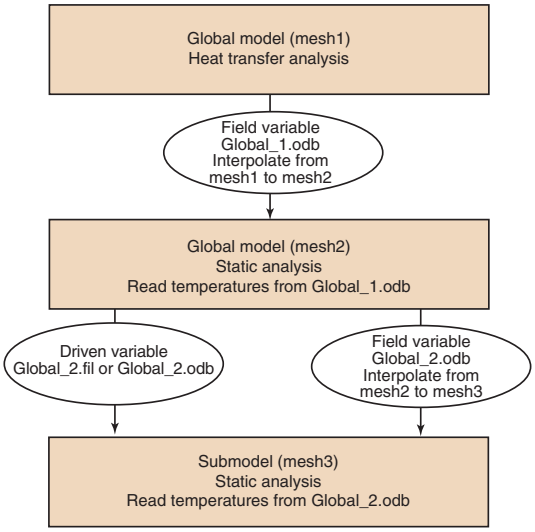


Figure 15: Sequentially coupled thermal-stress analysis for the global model with only a thermal-stress analysis for the submodel.

- Run a heat transfer analysis of the global model, and write the nodal temperatures to the results or output database file. Run a sequentially coupled thermal-stress analysis (the global thermal-stress analysis) using the same mesh (mesh1) as the global heat transfer analysis and the temperatures from the results or output database file for the global heat transfer analysis. Next, run a submodel heat transfer analysis using the mesh (mesh2) that is required for the final submodel thermal-stress analysis, and write the nodal temperatures to the results or output database file. Use the temperature solution from the global heat transfer analysis to drive the solution of the submodel heat transfer analysis. Finally, run the submodel thermal-stress analysis using the temperatures (as field variables) obtained from the results or output database file for the submodel heat transfer analysis and the displacements (as driven variables) obtained from the global thermal-stress analysis. See the detailed flow chart in *Figure 16*.

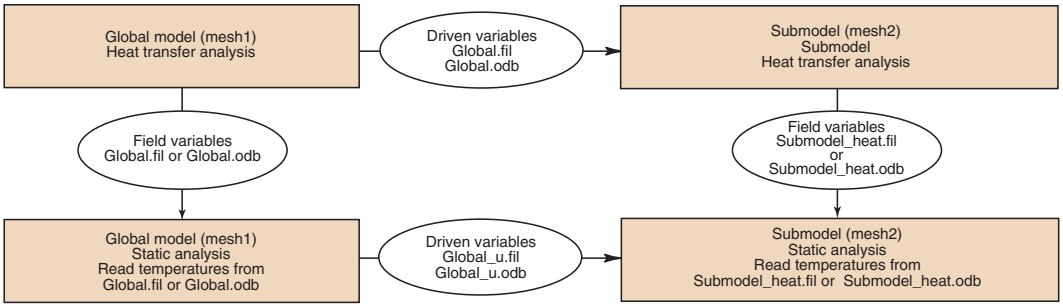


Figure 16: Sequentially coupled thermal-stress analysis for both the global model and submodel.

Material Options

Any of the material models described in *Abaqus Materials Guide* can be used in the global and submodel analyses. The material response defined for the submodel might be different from that defined for the global model.

Elements

The dimensionality of the submodel must be the same as that of the global model: both models must be either two-dimensional or three-dimensional. The following limitations apply:

- The boundary nodes of the submodel must lie within regions of the global model where Abaqus is able to perform spatial interpolation to define the values of the driven variables. Therefore, they must lie within (or, as allowed by the exterior tolerance, near to) two- or three-dimensional geometrically defined elements in the global model. Such geometrically defined elements are:
  - first- or second-order triangles or quadrilaterals in two dimensions;
  - first- or second-order triangular or quadrilateral shells;
  - first- or second-order triangular or quadrilateral membranes; and
  - first- or second-order tetrahedra, wedges, or bricks in three dimensions.
- When shell elements with five degrees of freedom per node (S4R5, S8R5, STRI65, etc.) are used in the global model, the rotations are not written to the results file or the output database; therefore, only the displacement degrees of freedom can be driven. This restriction suggests that submodeling should not be used with these elements or that the submodel should include a set of narrow elements around its driven edges so that the interpolated displacements at these nodes effectively transfer the rotation. Five degree of freedom shells cannot be used in shell-to-solid submodeling.
- The boundary nodes cannot lie in regions of the global model where there are only one-dimensional elements (beams, trusses, links, axisymmetric shells) since Abaqus does not provide the necessary interpolation of results for such elements.
- The boundary nodes cannot lie in regions of the global model where there are only user elements, substructures, springs, dashpots, etc. since those element types do not allow for geometric interpolation.
- The boundary nodes cannot lie in regions of the global model where there are only axisymmetric solid elements with nonlinear, asymmetric deformation (CAXA elements). The submodeling capability is currently not supported for these elements.
- The reference node associated with generalized plane strain elements (CPEG) cannot be used as a driven boundary node in a submodeling analysis.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers*).

As described above, nodal output requests to the results file or output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

Input File Template

Global Analysis:

```
*HEADING
…
*STEP
Step 1
*STATIC (or *DYNAMIC, etc.)
Data line to define step time and control incrementation
…
*NODE FILE
List of solution variables to be used to drive the submodel
*OUTPUT, FIELD
*NODE OUTPUT
List of solution variables to be used to drive the submodel
*END STEP
```

Submodel Analysis Using the Submodel Interface:

```
*HEADING
…
*SUBMODEL, EXTERIOR TOLERANCE=tolerance
List of all nodes to be driven
**
*STEP
*STATIC (or any other allowable procedure)
Data line to define step time (must be the same as the step time in the global analysis unless the
TIMESCALE parameter is used on the *BOUNDARY option) and control incrementation
…
*BOUNDARY, SUBMODEL, STEP=1
Data lines listing nodes and degrees of freedom to be driven in this step
…
*END STEP
```

Submodel Analysis Using the Field Import Interface:

```
*HEADING
…
*STEP
*STATIC
…
*FIELD IMPORT, SUBMODEL
*EXTERNAL FIELD, FILE=file_name.sim
Data lines to define external field
…
*END STEP
```

Surface-Based Submodeling

Products: Abaqus/Standard

References:

- *About Submodeling*
- *Output to the Output Database*
- **SUBMODEL*
- **DSLOAD*
- *Submodeling*

Overview

The surface-based submodeling technique:

- might not provide the same level of accuracy as node-based submodeling;
- should be used only when the node-based technique cannot provide adequate results;
- is limited to stress-based solid-to-solid submodeling for general static procedures (see *Static Stress Analysis*) in Abaqus/Standard;
- applies surface tractions to submodel surfaces based on a stress field interpolated from the global model; and
- can be combined with node-based submodeling of displacements (see *Node-Based Submodeling*).

Surface-Based Submodeling

Surface-based submodeling is provided as a complement to the node-based technique, enabling you to drive the submodel with stresses from the global model.

Input File Usage: **SUBMODEL*, TYPE=SURFACE

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Element Types Supported

The following types of submodeling are provided for the surface-based approach (global-to-submodel):

- Two-dimensional models: Solid-to-solid
- Three-dimensional models: Solid-to-solid

You can use different element types in the submodel than those used to model the corresponding region in the global model. In general, for surface-based submodeling Abaqus/Standard supports all continuum elements supported for the static analysis procedure; however, it does not support the following:

- Cylindrical elements
- Continuum shell elements
- Continuum solid elements with composite sections

Procedures Supported

You can use the surface-based technique only for static analysis in Abaqus/Standard.

Your submodel cannot refer to a global model step that includes multiple load cases (see *Multiple Linear Load Case Analysis*). You must perform the global analysis with a single load definition for the step that will drive the submodel.

Performing a Surface-Based Submodeling Analysis

Your submodel analysis is driven, either partly or completely, from the results obtained from a global model analysis. The results from the global model are interpolated onto the surfaces on the appropriate parts of the boundary of the submodel. Thus, the response at the boundary of the local region is defined by the solution for the global model. The driven surfaces and any loads applied to the local region determine the solution in the submodel.

Surface-based submodeling should be used only when the node-based technique cannot provide adequate results. For a comparison of the two submodeling techniques and recommendations for their application, refer to *About Submodeling*.

Saving the Results from the Global Model

The results from the global analysis must be saved at all elements required for the interpolation of the driven variables to the boundary surface of the submodel. Only the output database (in ODB or SIM format) can be used for this purpose.

In each step of the global model whose solution will be used to drive the submodel, write the stress results to the output database (see *Output to the Output Database*).

Input File Usage: Use both of the following options:

**OUTPUT*, FIELD
**ELEMENT OUTPUT*

Abaqus/CAE Usage: Step module: **Output->Field Output Requests->Create**

Referring to the Global Model Results from the Submodel Analysis

You must define the source of the global solution results and provide the name of the output database file (in ODB or SIM format); the file extension is optional. If the file extension is omitted, Abaqus will use in order, the ODB output database file or the SIM database file.

Input File Usage: **abaqusjob=***submodel_input_file* **globalmodel=** *global_output_database* or *sim_database_file*

Abaqus/CAE Usage: Any module: **Model->Edit Attributes->*submodel*: Submodel**: **Read data from job**: *global_output_database*

Reading data from a SIM database file is not supported in Abaqus/CAE.

Specifying the Global Elements Used to Drive the Submodel

To preclude certain elements from driving the submodel, you can specify a global element set to limit the search to an appropriate subset of the global model (see *Specifying the Global Elements Used to Drive the Submodel*).

Specifying the Driven Surfaces in the Submodel

Specifying the driven element-based surfaces does not activate the driven surface loads: they must be activated by specifying the appropriate submodel distributed surface loads.

All surface facets of the submodel to be driven by stresses in any step must be specified as driven surfaces since the list of surfaces cannot be extended subsequent to its initial definition (even at restart). However, variables at the surfaces given do not have to be driven in all steps: the choice of which surfaces are driven in a particular step is made as part of a submodel distributed surface load definition, as discussed in *Defining the Driven Surface Tractions in the Submodel* later in this section.

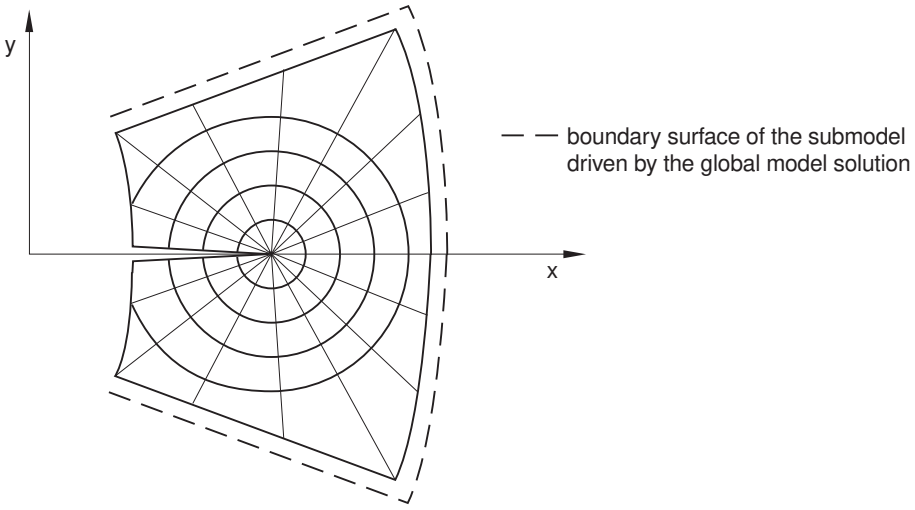


Figure 1: The magnified submodel.

Input File Usage: **SUBMODEL*
list of element-based structural surfaces

The **SUBMODEL* option must be included in the model definition portion of the input file for the submodel analysis. Multiple **SUBMODEL* options are allowed; however, in this case you must ensure that the driven surfaces specified on the data line of one option are separate and distinct from the other surfaces specified on the data lines of all the other options.

Abaqus/CAE Usage: Load module: **Create Load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: select region

Defining Geometric Tolerances

A geometric tolerance is used to define how far driven element-based surface nodes in the submodel can lie outside the exterior surface of the global model, as that surface is interpolated in the global, undeformed finite element model. By default, surface nodes in the submodel must lie within a distance calculated by multiplying the average element size in the global model by 0.05. You can change the tolerance, which is useful in cases where submodel driven surfaces lie to a greater extent outside the global model exterior surface. Tolerances

larger than this default value, however, can result in significantly greater computation times and lower accuracy in the driven solution for driven surface regions significantly outside the global model exterior surface.

You can define the geometric tolerance as a fraction of the size of the average element in the global model or as an absolute distance in the length units chosen for the model. If both tolerances are defined, Abaqus uses the tighter tolerance.

Input File Usage: Use the following option to define the geometric tolerance as an absolute distance:

**SUBMODEL*, TYPE=SURFACE, ABSOLUTE EXTERIOR
TOLERANCE=*tolerance*

Use the following option to define the geometric tolerance as a fraction of the size of the average element in the global model:

**SUBMODEL*, TYPE=SURFACE, EXTERIOR TOLERANCE=*tolerance*

Abaqus/CAE Usage: Load module: **Create Load** choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Exterior tolerance: absolute:** or **relative:** *tolerance*

The Exterior Tolerance in Solid-to-Solid Submodeling

The exterior tolerance for a solid-to-solid submodel analysis is indicated by the shaded region in *Figure 2*. If the distance between the driven surface nodes and the free surface of the global model falls within the specified tolerance, the solution variables from the global model are extrapolated to the submodel.

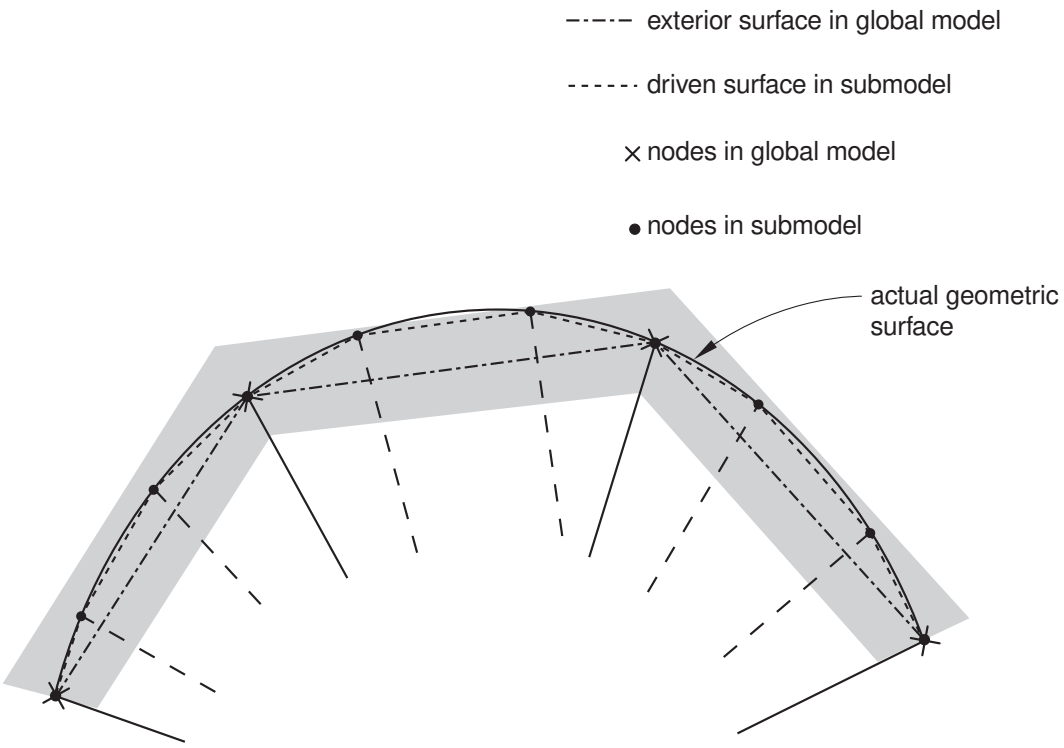


Figure 2: The exterior tolerance in surface-based submodeling.

Defining the Driven Surface Tractions in the Submodel

The actual driven surface tractions are defined in any step as submodel distributed surface loads. The stresses resulting in these tractions are “driven variables” obtained from the output database file of the global analysis.

All stress components from the global model elements that will drive the submodel boundary surface must have been written to the output database. They will be used to create traction, shear, and normal stresses at integration points of driven surfaces (as nonuniform distributed surface loads). All applicable stress components are calculated and applied to the surface integration points at each time increment.

Input File Usage: **DSLOAD*, SUBMODEL

Abaqus/CAE Usage: Load module: **Create load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**

Specifying the Step Number from the Global Analysis

You specify the step of the global model history that is to be used for the driven variables in the current submodel analysis step.

Input File Usage: **DSLOAD*, SUBMODEL, STEP=*step*

Abaqus/CAE Usage: Load module: **Create load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number:** *step*

Modifying the Set of Driven Surface Tractions

You can modify the submodel distributed surface load definitions from step to step to change the global step reference, you can remove surface load definitions, and you can reintroduce them later (see *About Loads*). Submodel distributed surface loads do not propagate between steps. At each new step all submodel distributed surface loads defined in previous steps will be removed unless they are modified or redefined. New surfaces cannot be added to the total set of driven surface defined for the submodel; this set of driven surfaces is a fixed part of the model definition.

Input File Usage: Use one of the following options:

**DSLOAD*, SUBMODEL, OP=MOD
**DSLOAD*, SUBMODEL, OP=NEW

Guidelines for Obtaining Adequate Solution Accuracy

Unlike node-based submodeling, surface-based submodeling can in many cases provide incorrect or misleading submodel results. This risk follows from the methods used to interpolate stresses from the global model to the submodel:

- The global model material point stresses are smoothed and associated with the global model nodes.
- These global model node-located stresses are then interpolated to the submodel surface integration points and applied as tractions.

This process is generally nonconservative, resulting in a submodel traction field that is not equivalent to the global model stress field in an equilibrium sense.

Modeling Guidelines

You can improve accuracy and achieve reasonable submodel solutions by observing the following guidelines:

- Design your models so that your submodel surface intersects the global model in regions of relatively low stress gradients.
- Design your models so that your submodel surface intersects the global model in regions of uniform element size. A warning message is provided in the data (`.dat`) file in cases where significant nonuniform element size distributions are seen.

Checking Your Results

To understand whether your modeling approach results in a reasonably accurate solution, the following guidelines are recommended:

- Compare the stress distributions on the submodel-driven surfaces with the stress distributions in the global model. You can view the stress distributions in the global model by using tools such as cutting planes and path plots in the Visualization module of Abaqus/CAE. The degree to which the global model's stress distributions agree with those in the submodel-driven surface is generally an indication of the level of accuracy of your submodel solution.
- When using inertia relief in the submodel for cases where submodeling does not remove all rigid body modes, compare the inertia relief forces to the prevailing force level in your submodel. If the inertia relief force is large compared to the prevailing force level, your submodel results might be inaccurate.

Special Considerations

There are several special considerations that are worth noting.

Handling of Rigid-Body Modes

When you use surface-based submodeling exclusively to drive your submodel response, your displacement solution will not be unique; you will generally encounter rigid-body modes and accompanying numerical issues. You can address these rigid-body modes by

- providing sufficient node-based submodel displacement boundary condition definitions in the submodel analysis,
- providing sufficient boundary condition definitions in the submodel analysis, or
- providing an inertia relief load definition in the submodel analysis (see *Inertia Relief*).

You can combine these definitions, as necessary and appropriate to your model, to address all rigid body modes.

Cases of Finite Rotation

Global model stress results are stored in the output database in the global coordinate system. Submodel tractions are calculated from these stresses and the current configuration surface normal in the submodel. Hence, when your global model result involves significant finite rotation, your submodel results will generally be inaccurate unless you provide sufficient node-based submodel displacement boundary condition definitions to impart similar rigid-body rotations to the submodel; exclusive use of surface-based submodeling definitions is not adequate to provide these rigid-body motions. You might also experience convergence difficulties in the submodel when it is not properly rotated.

Inelastic Behavior

When surface-based submodeling is used to drive a submodel region with an inelastic material definition, you might encounter rigid-body modes and accompanying numerical issues. For example, numerical issues will prevent convergence if the submodel material definition includes plasticity and the submodel loading results in a shear band formation beyond the material hardening definition, such that unconstrained motion can occur (that is, if the submodel loads exceed the limit load capacity). In these cases node-based submodeling should be used.

Procedures

Only the static procedure is allowed. Both general (possibly nonlinear) and linear perturbation steps can be used in submodeling (see *General and Perturbation Procedures* for a discussion of general and linear perturbation steps).

Obtaining a Solution at a Particular Point in Time Using Linear Perturbation Analysis

In Abaqus/Standard it is possible to study the submodel's linearized response corresponding to a particular point in time in the global solution by using a static, linear perturbation procedure in the submodel analysis. You can select the increment in the global analysis step that is to be used as the basis for calculating the values for the driven variables. If you do not select an increment in a static linear perturbation step, the last increment of the selected step in the global analysis is used as the basis for calculating the values for the driven variables. You cannot select an increment in a general submodel step.

Input File Usage: **DSLOAD*, SUBMODEL, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Selection of a specific global model increment is not supported in Abaqus/CAE.

Mixing General and Linear Perturbation Steps

It is possible to mix general steps and linear perturbation steps in both the global and the submodel analyses. Abaqus allows general analysis steps to be treated as linear perturbation steps during submodeling, and vice versa.

Example: Submodeling with General and Linear Perturbation Steps

For an example of submodeling that uses both general and linear perturbation steps, consider the following situation. The global analysis consists of a static preload—done as a general, nonlinear, analysis step—followed by extraction of the eigenmodes of the preloaded structure, then a step of 5 seconds of modal dynamic response analysis:

```
*STEP
** Apply preload
*STATIC
 0.1, 1.0
…
** Write out stress results for elements needed to
** interpolate to the submodel's surfaces
*ELEMENT OUTPUT, ELSET=DETAIL
 S
*END STEP
*STEP
** Calculate modes and frequencies
*FREQUENCY
…
** The *ELEMENT OUTPUT option is repeated because
```

```
** this is the first linear perturbation step
*ELEMENT OUTPUT, ELSET=DETAIL
 U
*END STEP
*STEP
** Dynamic response of preloaded system
*MODAL DYNAMIC
 0.01, 5.0
…
*END STEP
```

We wish to study the local, possibly nonlinear, response of a part of this model that is so small that we do not need to model dynamic effects locally and can, thus, perform two steps of static analysis:

```
** Define submodel surfaces (driven surfaces)
*SUBMODEL,TYPE=SURFACE
PERIM
*STEP
** Preload
*STATIC
 0.1, 1.0
*DSLOAD, SUBMODEL, STEP=1
…
*END STEP
*STEP
** Local static response to global dynamic step
*STATIC
 0.01, 5.0
*DSLOAD, SUBMODEL, STEP=3
…
*END STEP
```

It is perfectly acceptable that the submodel analysis requests general, possibly nonlinear, analysis for both steps, while in the global analysis the dynamic step was a linear perturbation step (modal dynamics is always a linear perturbation analysis). It is your responsibility to judge that this use of the submodeling feature is reasonable. For example, suppose that the global analysis were continued with a fourth step of general, nonlinear static response:

```
*RESTART, READ, STEP=3
** Read state at end of initial preload
** (could equally well use *RESTART, READ, STEP=1)
*STEP
** Add more preload
*STATIC
 0.2, 1.0
…
*END STEP
```

This fourth general analysis step starts with the state at the end of general analysis Step 1 because the frequency extraction and the modal dynamic steps are both linear perturbation steps. However, if we restart the submodel analysis in the same way, the solution might not be comparable with the global model solution:

```
*RESTART, READ, STEP=2
** Read state at end of step 2
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*DSLOAD, SUBMODEL, STEP=4
…
*END STEP
```

The second step in the submodel is a general analysis step, to which the response might be nonlinear, thus changing the state of the model. A valid alternative would be to apply the Step 4 response to the submodel immediately after the first step:

```
*RESTART, READ, STEP=1
** Read state at end of preload step
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*DSLOAD, SUBMODEL, STEP=4
…
*END STEP
```

Loads

Any loads that are applied in the submodel region of the global analysis must be imposed in the submodel analysis in the usual way. It is your responsibility to apply such loads to the submodel correctly so that they correspond to the loading of the global model. See *About Loads* for an overview of the loads available in Abaqus.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see *Abaqus/Standard Output Variable Identifiers* and *Abaqus/Explicit Output Variable Identifiers*).

As described above, element stress output requests to the output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

Input File Template

Global Analysis:

```
*HEADING
…
*STEP
Step 1
*STATIC (or *STATIC, etc.)
Data line to define step time and control incrementation
…
*ELEMENT OUTPUT
S
*OUTPUT, FIELD
*ELEMENT OUTPUT
S
*END STEP
```

Submodel Analysis:

```
*HEADING
…
*SUBMODEL,TYPE=SURFACE, EXTERIOR TOLERANCE=tolerance
List of all surfaces to be driven
**
*STEP
*STATIC (or any other allowable procedure)
Data line to define step time and control incrementation.
…*DSLOAD, SUBMODEL, STEP=1
Data lines listing surfaces to be driven in this step
```

```
…
*END STEP
```

Matrices

In this section:

- *Using Matrices*
- *Generating Matrices as a Linear Analysis Step*
- *Generating Matrices from Nonlinear Procedures*

Using Matrices

Products: Abaqus/Standard

References:

- *Generating Matrices as a Linear Analysis Step*
- **MATRIX ASSEMBLE*
- **MATRIX GENERATE*
- **MATRIX INPUT*
- **MATRIX OUTPUT*

Overview

A matrix:

- can be used to represent stiffness, mass, viscous damping, or structural damping for a part of the model or for the entire model;
- is defined by giving it a unique name and by specifying matrix data, which may be scaled;
- can be symmetric, unsymmetric, or have both instances;
- can be given in text format in lower triangular, upper triangular, or square form or read from binary `.sim` files generated by the matrix generation procedure;
- can be used to provide linear elastic response with large translations but not large rotations;
- can be used in static and natural frequency extraction procedures;
- can be used in direct-integration implicit dynamic analysis;
- can be used in matrix generation and substructure generation procedures;
- can be used in direct steady-state dynamic analysis;
- can be used in transient modal dynamics, mode-based steady-state dynamics, subspace-based steady-state dynamics, random response, response spectrum, and complex eigenvalue extraction procedures that use the SIM architecture;
- can have loads, boundary conditions, and constraints applied directly to any matrix nodal degrees of freedom;
- can be used in submodeling analysis; and
- cannot be used in mode-based analyses that do not use the SIM architecture.

What Is a Matrix in Abaqus/Standard?

Designing complex models of structures like automobiles typically involves subcontracting the work on various parts. When the entire model has to be put together, information about the parts needs to be exchanged between different vendors. Often, to avoid the disclosure of proprietary information, this information is exchanged in terms of matrices representing the stiffness, mass, and damping for each part. During an analysis these matrices are added to the corresponding global finite element matrices to complete the assembly of the entire model.

Abaqus/Standard provides the capability to input stiffness, mass, viscous damping, and structural damping matrices directly. You can define as many different matrices as are necessary to build the model.

Including Matrices in a Model

You must assign a name to the matrix to include it in the matrix usage model.

Input File Usage: **MATRIX INPUT*, NAME=*name*

Specifying a Matrix Type

For matrices given in text format, you can specify the matrix type as symmetric (default) or unsymmetric. If symmetric, it can be entered as a lower triangular, upper triangular, or square matrix.

For matrices read from a `.sim` file, Abaqus/Standard sets the matrix type automatically according to the matrix data stored on the SIM database, and any user-specified value is ignored.

You can also provide both instances, symmetric and unsymmetric, for the same matrix. The relevant matrix instance is selected according to the analysis procedure. This approach is typically used for generally unsymmetric analyses containing inherently symmetric procedures such as *Natural Frequency Extraction*.

Input File Usage: Use one of the following options to specify the type for matrices given in text format, or use both options to include both instances of the matrix:

**MATRIX INPUT*, NAME=*name*, TYPE=UNSYMMETRIC

**MATRIX INPUT*, NAME=*name*, TYPE=SYMMETRIC

Scaling the Matrix Data

You can define a multiplication scale factor for all matrix entries.

Input File Usage: **MATRIX INPUT*, NAME=*name*, SCALE FACTOR=*sval*

Providing Matrix Data Directly

You can specify data directly to define a symmetric matrix in lower triangular, upper triangular, or square format. For a square matrix to be symmetric, corresponding entries above and below the diagonal must have exactly the same values. When you specify several matrices for the same operator, they must all be of the same format: lower triangular, upper triangular, or square.

You can specify data directly to define an unsymmetric matrix by providing data for each matrix entry. If an entry above or below the diagonal of an unsymmetric matrix is specified explicitly, the corresponding entry below or above the diagonal must also be specified even if that value is zero.

Input File Usage: **MATRIX INPUT*
row node label, *degree of freedom for row node*, *column node label*, *degree of freedom for column node*, *matrix entry*

Repeat this data line to specify data for each matrix entry.

Reading the Matrix Data in Text Format from an Alternate File

Matrix data in text format can be contained in an alternate file. Typically, an alternate file is used for large matrices. To ensure acceptable performance, the data lines in the alternate file are read without extensive checking for data format. You should make sure that the data entries are specified in the proper format without any

comments or blank lines. Matrix data output in text format can be generated in the matrix generation procedure (see *Output*).

Input File Usage: **MATRIX INPUT*, NAME=*name*, INPUT=*input_file_name*

Reading the Matrix Data from the SIM Database

Matrix data in binary format can be read from the `.sim` file generated by the matrix generation procedure (see *Matrix Generation*). The `.sim` file can contain stiffness, mass, viscous damping, and structural damping matrices. You specify each matrix to be read from the `.sim` file.

Input File Usage: Use the following options:

**MATRIX INPUT*, NAME=*stif_name*, INPUT=*sim_file_name*, MATRIX=STIFFNESS

**MATRIX INPUT*, NAME=*mass_name*, INPUT=*sim_file_name*, MATRIX=MASS

**MATRIX INPUT*, NAME=*dmpv_name*, INPUT=*sim_file_name*, MATRIX=VISCOUS DAMPING

**MATRIX INPUT*, NAME=*dmps_name*, INPUT=*sim_file_name*, MATRIX=STRUCTURAL DAMPING

Internal Nodes in Matrix Data

Internal nodes are nodes with internal degrees of freedom associated with them (for example, Lagrange multipliers or generalized displacements). Abaqus/Standard must distinguish between the user-defined and internal nodes and must distinguish between the internal node types to solve the problem correctly. Matrices generated in the binary SIM format contain all of the necessary information about the internal nodes. For matrices generated in the text format, by definition, user-defined nodes have positive node labels, and internal nodes have negative node labels.

By default, you cannot include a matrix including entries associated with internal nodes in the model. However, you can override this restriction by specifying the internal degree of freedom type. You can specify that internal degrees of freedom represent Lagrange multipliers or represent substructure generalized degrees of freedom associated with the substructure dynamic modes.

In addition, Abaqus applies the specified internal degree of freedom type to all internal degrees of freedom associated with the matrix entries. Therefore, the text format has significant limitations if the matrix data contain references to internal nodes. For this case, it is strongly recommended that you generate and use matrices in the binary SIM format (see *Generating Matrices as a Linear Analysis Step*).

Input File Usage: Use the following option to specify that internal degrees of freedom represent Lagrange multipliers:

**MATRIX INPUT*, INTERNAL DOFS=LAGRANGE

Use the following option to specify that internal degrees of freedom represent substructure generalized degrees of freedom associated with the substructure dynamic modes:

**MATRIX INPUT*, INTERNAL DOFS=MODAL

Defining the Stiffness, Mass, and Damping with Matrices Included in a Model

You can assemble the stiffness, mass, viscous damping, and structural damping matrices that you have specified into the corresponding global finite element matrices for the usage level model. Many matrices with different names can be defined and assembled.

Input File Usage: Use the following option to assemble matrices generated from the same original model:

**MATRIX ASSEMBLE*, STIFFNESS=*stif_name*, MASS=*mass_name*, VISCOUS DAMPING=*dmpv_name*,
STRUCTURAL DAMPING=*dmps_name*

Each **MATRIX ASSEMBLE* option defines a matrix subassembly. The stiffness, mass, and damping matrices included in the subassembly must be generated from the same original model or defined on the same set of nodes.

To assemble matrices generated from different original models, repeat the **MATRIX ASSEMBLE* option for each model.

Connecting a Part of a Model Represented by Matrices

You can connect a part of the model represented by user-defined matrices to other parts of the model through shared nodes. You must define these nodes directly in the model (see *Node Definition*). In addition, there may be matrix nodes that are not shared with the finite elements of the global model. You do not need to define matrix nodes that have no loads, boundary conditions, or constraints associated with them. Abaqus automatically defines these nodes at the origin of the global coordinate system for matrices specified in the text format or in the geometrical positions defined by the nodal coordinates of the original model and applied positioning transformations for matrices specified in the binary format in the `.sim` file. If the geometrical positions of the matrix nodes are not correct (as for automatically generated nodes of the matrices specified in the text format), Abaqus may calculate incorrect results for the global inertia tensor of the usage model, modal participation factors, and some other quantities. You should exercise caution when using matrices specified in the text format.

Input File Usage: Use the following option to define the shared nodes directly:

**NODE*

Translating, Rotating, and Reflecting a Matrix Subassembly

You can specify translations, rotations, and/or reflections of a matrix subassembly in a matrix assemble definition. You specify a translation by giving a translation vector. You specify a rotation by giving two points that define a rotation axis and a right-handed angular rotation about that axis. You specify a reflection by giving three noncollinear points in the reflection plane. You can also define a position transformation of a matrix subassembly by specifying three successive rotations with respect to a reference point using the Euler angles.

Positioning transformations are applied in the order they are specified. There is no restriction on the number of translations, rotations, and reflections for a single matrix assemble definition. When a matrix subassembly is translated, rotated, and/or reflected, the degrees of freedom at any matrix nodes are defined with respect to the coordinate directions at the usage level.

If a subassembly including matrices specified in the binary format has position transformations, automatically generated matrix nodes are placed in the geometrical positions associated with the transformed nodal coordinates.

Setting a Tolerance on the Matrix Subassembly Nodes

One difficulty with using matrices is ensuring that the subassembly nodes are connected to the correct nodes on the usage level (after positioning translation, rotation, and/or reflection, if applicable). Therefore, Abaqus/Standard checks that the coordinates of the matrix nodes match the coordinates of the corresponding nodes on the usage level.

To check the coordinates, you can set a tolerance on the distance between usage level nodes and the corresponding matrix nodes. This tolerance indicates the largest deviation allowable before a warning is issued. If you do not specify this tolerance, the default is to use a tolerance of 10^{-4} times the largest estimated characteristic dimension within the matrix subassembly. If you specify a tolerance of 0.0, the position of the matrix nodes is not checked.

The geometric check uses the coordinates of the matrix nodes after translation, rotation, and/or reflection of the matrix subassembly at the usage level.

Input File Usage: **MATRIX ASSEMBLE*, POSITION TOL=*tolerance*

Remapping User-Defined Nodes in Assembled Matrices

The nodes defined in the assembled matrices can be remapped (renamed) to different node labels in the matrix usage model. You must define all the new node labels in the matrix usage model, create a node set from them, and specify this node set when assembling the matrices. The size of the node set and the order of the nodes in the set must fully correspond to the combined set of nodes of all the matrices that are assembled. The matrix nodes are assumed to be sorted in ascending order of their original labels that were defined at generation or specified in the matrix data.

Input File Usage: Use the following option to create a node set for the matrix nodes:

**NSET*, NSET=*nset_name*, UNSORTED

Use the following option to assemble matrices with node remapping:

**MATRIX ASSEMBLE*, STIFFNESS=*stif_name*, MASS=*mass_name*, VISCOUS DAMPING=*dmpv_name*,
STRUCTURAL DAMPING=*dmps_name*, NSET=*nset_name*

Multiple Instantiation of Matrices

With the node remapping feature, the same matrix can be used multiple times in the matrix usage model. You define the matrix once and assemble it several times, specifying the relevant node sets for remapping.

Input File Usage: **MATRIX INPUT*, NAME=*name*
**MATRIX ASSEMBLE*, STIFFNESS=*name*
**MATRIX ASSEMBLE*, STIFFNESS=*name*, NSET=*nset1_name*
**MATRIX ASSEMBLE*, STIFFNESS=*name*, NSET=*nset2_name*

Remapping Internal Nodes in Assembled Matrices

When using matrix data that contain internal nodes, Abaqus automatically maps those nodes to unique internal node labels in the matrix usage model. Assembled stiffness, mass, and damping matrices that originate from the same finite element model and represent the same part of the usage model must share the internal nodes. It is important to assemble such stiffness, mass, and damping matrices together using a single matrix subassembly definition.

For assembled matrices that originate from different models or for different instances of a matrix subassembly representing different parts of the usage model, the internal nodes are mapped to different internal nodes in the matrix usage model, even if they have the same negative node labels in the text format. You must use different matrix subassembly definitions to define different matrix subassemblies or different instances of a matrix subassembly.

Remapping Interface Nodes in Assembled Matrices

You can use interface nodes of the matrices specified in the binary SIM format to connect a matrix subassembly to the matrix usage model. To connect a matrix subassembly, you can define the new node labels for the interface nodes in the matrix usage model, create a node set from them, and specify this node set when assembling the matrices. The size of the node set and the order of the nodes in the set must fully correspond to the set of interface nodes defined in the matrix generation analysis. You do not need to define nodes that are not interface nodes. Abaqus automatically creates these nodes in the geometrical positions defined by the nodal coordinates of the original model and applied positioning transformations.

Input File Usage: Use the following option to create a node set for the matrix nodes:

**NSET*, NSET=*nset_name*, UNSORTED

Use the following option to assemble matrices with node remapping:

**MATRIX ASSEMBLE*, STIFFNESS=*stif_name*, MASS=*mass_name*, VISCOUS DAMPING=*dmpv_name*,
STRUCTURAL DAMPING=*dmps_name*, NSET=*nset_name*

Specifying the Matrix Subassembly Name

You can specify the matrix subassembly name to reference the subassembly if it is required in the modeling features.

Input File Usage: **MATRIX ASSEMBLE*, NAME=*asmbf_name*

Mapping Matrix Subassembly Nodes to the Usage Model Nodes

Abaqus automatically generates the mapping between the matrix subassembly nodes and the usage model nodes. This mapping is written in the text format to the `.mtxmap` file. The file contains the subassembly name, node number, and mapping pairs: original label, new label. This information is repeated for each subassembly with automatically mapped nodes.

Using Matrices in Nonlinear Analyses

When you use matrices in a nonlinear analysis procedure, nonlinearities are not accounted for. Since the matrix data remain unchanged during the analysis, only linear elastic material behavior can be represented and only large translations can be modeled correctly in a geometrically nonlinear analysis. Changes to the matrix due to large rotations or load stiffness are not computed in a geometrically nonlinear analysis.

Using Matrices in Linear Perturbation Analyses

Matrices can be used in a static perturbation analysis as well as in a natural frequency extraction analysis using the Lanczos or AMS eigensolver. For certain quantities (such as participation factors and global inertia properties) to be computed properly, the coordinates of the nodes associated with the matrices should be defined in the model using matrices. Matrices can also be used in modal analysis procedures using the high-performance SIM architecture; namely, steady-state dynamic, modal dynamic, random response, response spectrum, and complex

frequency extraction analyses. Matrices can be used in the substructure generation and matrix generation procedures as well.

Matrices cannot be used in the modal procedures that are not based on the high-performance SIM architecture.

Constraints and Transformations

Kinematic constraints (for example, coupling constraints, linear constraint equations, multi-point constraints, or surface-based tie constraints) can be applied to any nodes in a model containing matrices. Since kinematic constraints in Abaqus/Standard are usually imposed by eliminating degrees of freedom at the dependent nodes, matrix nodes should not be used as dependent nodes.

To apply contact constraints on matrix nodes, a node-based surface must be defined on these nodes and this surface should be used as the secondary surface in the contact pair definition.

Nodal transformations defined at nodes that appear in the matrix do not affect the matrix. The matrix entries corresponding to these nodes are assumed to be in the local coordinates defined by the nodal transformations.

Initial Conditions

Initial conditions can be specified as usual; however, only node-based initial conditions can be applied to nodes that appear in matrices. See *Initial Conditions*.

Boundary Conditions

Boundary conditions can be specified as usual. See *Boundary Conditions*. Any boundary conditions that are defined in the matrix generation step are implied in the matrices and should not be reapplied in the matrix usage analysis.

Matrix nodes can be defined as driven nodes in a submodel analysis (see *About Submodeling*); they cannot be defined as driving nodes in a global model. For shell-to-solid submodeling, matrix nodes that are defined as driven nodes are treated as lying within the center zone no matter how far they are from the shell reference surface.

Loads

Concentrated nodal forces can be applied at displacement degrees of freedom (1–6) of any node as usual. Distributed pressure forces can be applied to surface elements defined over matrix nodes (see *Surface Elements*). Body forces cannot be applied to parts of the model represented by matrices. User-defined loads can be applied with the same restrictions as above for distributed pressure forces and body forces.

Predefined fields can be applied at any nodes as usual (see *Predefined Field Variables* and *Predefined Temperature*); however, matrix data are not affected by predefined fields. For example, if temperatures are specified as a predefined field on nodes that appear on a matrix, only the elements that share these nodes with the matrix experience thermal strains if thermal expansion is specified for those elements. The matrix does not experience any thermal strains, but it may experience linear elastic forces due to displacements at shared nodes.

Output

All nodal output variables that apply to the analysis are available (see *Abaqus/Standard Output Variable Identifiers*).

Limitations

The following are known limitations to using matrices:

- Matrices cannot be used in a model containing parts and assemblies.

- Matrices containing acoustic pressure and mechanical degrees of freedom will disable the coupled acoustic structural eigenvalue extraction.
- By default, using the matrix data containing internal nodes in text format is not supported. Usage of such matrices in text format can be allowed for some special cases. This feature should be used with caution.
- Matrices cannot be used in nonlinear analyses with inertia relief loads.
- Matrices cannot be used in linear perturbation analyses with inertia relief loads using distributed memory parallel (DMP) execution.
- Matrices cannot be used in mode-based analyses that do not use the new linear dynamics architecture (SIM-based architecture).
- Matrices cannot be used in nonlinear analyses using the quasi-Newton solution technique.

Input File Template

```
*HEADING
…
*NODE
Data lines to specify nodes
*NSET, NSET=NSET1, UNSORTED
Data lines to specify a node set with the nodes in a particular order
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*MATRIX INPUT, NAME=MAT1, SCALE FACTOR=sval
Data lines to specify a  stiffness matrix
*MATRIX INPUT, NAME=MAT2, SCALE FACTOR=sval
Data lines to specify a  mass matrix
*MATRIX INPUT, NAME=MAT3, SCALE FACTOR=sval
Data lines to specify a  viscous damping matrix
*MATRIX INPUT, NAME=MAT4, INPUT=input_file_name
*MATRIX INPUT, NAME=MAT5, INPUT=input_file_name
*MATRIX INPUT, NAME=MAT6, INPUT=sim_file_name, MATRIX=STIFFNESS
*MATRIX ASSEMBLE, STIFFNESS=MAT1, MASS=MAT2,
VISCOUS DAMPING=MAT3, STRUCTURAL DAMPING=MAT4
*MATRIX ASSEMBLE, STIFFNESS=MAT6, MASS=MAT5
*MATRIX ASSEMBLE, STIFFNESS=MAT6, MASS=MAT5, NSET=NSET1
*STEP(,NLGEOM)(,PERTURBATION)
Use NLGEOM  to include nonlinear geometric effects; it will remain active in all subsequent steps.
*STATIC
*BOUNDARY
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loads
*END STEP
*STEP
*FREQUENCY
*BOUNDARY
Data lines to prescribe zero-valued or nonzero boundary conditions
*END STEP
*STEP
*STEADY STATE DYNAMICS
*CLOAD and/or *DLOAD
Data lines to specify loads
*END STEP
```

Generating Matrices as a Linear Analysis Step

Products: Abaqus/Standard

Overview

Matrix generation:

- is a linear perturbation procedure;
- allows for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the stiffness, mass, viscous damping, structural damping, and load vectors in a model;
- generally creates matrices identical to those used in a subspace-based steady-state dynamic procedure (see *Subspace-Based Steady-State Dynamic Analysis*).
- includes initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the analysis;
- writes matrix data to a binary `.sim` file that can be read as input by Abaqus;
- can output matrix data to text files that can be read as input in other analyses by Abaqus or other simulation software; and
- can check the quality of the generated global stiffness and mass matrices.

References:

- *Using Matrices*
- *Element Matrix Assembly Utility*
- *Defining an Analysis*
- **MATRIX GENERATE*
- **MATRIX CHECK*
- **MATRIX OUTPUT*
- **MATRIX INPUT*
- **CLOAD*

Matrix Generation

Introduction

A linearized finite element model can be summarized in terms of matrices representing the stiffness, mass, damping, and loads in the model. Using these matrices, you can exchange model data between other users, vendors, or software packages without exchanging mesh or material data. Matrix representations of a model prevent the transfer of proprietary information and minimize the need for data manipulation.

The matrix generation procedure is a linear perturbation step (see *General and Perturbation Procedures*) that accounts for all current boundary conditions, loads, and material response in a model. You can also specify new boundary conditions, loads, and predefined fields within the matrix generation step. The generated matrices are input to a matrix usage model.

The matrix generation procedure uses SIM, which is a high-performance database available in Abaqus. The generated matrices are stored in a file named *jobname*_X*n*.sim, where *jobname* is the name of the input file or analysis job and *n* is the number of the Abaqus step that generates the matrices.

Specifying the Matrix Type

You can generate matrices representing the following model features:

- stiffness,
- mass,
- viscous damping,
- structural damping, and
- loads.

The load matrix contains integrated nodal load vectors (right-hand sides) for the load cases defined in the matrix generation step. Load cases can be made up of any combination of loadings—distributed loads, concentrated nodal loads, thermal expansion, and load cases defined for any substructures that may be used as part of the model.

When you generate the mass matrix with the mass only at the user-defined nodes, if a finite element has nonzero mass at Abaqus internal nodes, the mass is transferred to the user-defined nodes using heuristic techniques. Mass is redistributed only for modified triangle and tetrahedral elements; no mass is redistributed from internal to user-defined nodes for frame elements and substructures.

Input File Usage: Use the following option to generate the stiffness matrix:

**MATRIX GENERATE*, STIFFNESS

Use one of the following options to generate the consistent mass matrix:

**MATRIX GENERATE*, MASS

**MATRIX GENERATE*, MASS=CONSISTENT

Use the following option to generate the consistent mass matrix with mass only at the user-defined nodes:

**MATRIX GENERATE*, MASS=AT USER NODES

Use the following option to generate the viscous damping matrix:

**MATRIX GENERATE*, VISCOUS DAMPING

Use the following option to generate the structural damping matrix:

**MATRIX GENERATE*, STRUCTURAL DAMPING

Use the following option to generate the load matrix:

**MATRIX GENERATE*, LOAD

Friction Damping Effects

Friction at the contact nodes at which a velocity differential is imposed can give rise to the viscous damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces

stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with the velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see *Coulomb friction*. You can include these friction-induced contributions to the generated viscous damping matrix.

Input File Usage: **MATRIX GENERATE*, VISCOUS DAMPING, FRICTION DAMPING=YES

Generating Unsymmetric Matrices

Usually, the finite element operators (matrices) are symmetric, but the stiffness and damping matrices can be unsymmetric for a number of special modeling cases. For example:

- The stiffness matrix is unsymmetric for a model including sliding contact with friction.
- The viscous damping matrix is unsymmetric if it is generated for a steady-state-rotating model, such as a rolling tire.
- The viscous damping matrix is unsymmetric if friction-induced contributions are included.
- Unsymmetric matrices can be added to the model by specifying the matrix input directly, or they can be introduced by unsymmetric substructures.

Abaqus does not automatically generate unsymmetric matrices in these and other cases. You must explicitly select the unsymmetric solver (see *Defining an Analysis*) for the matrix generation step to obtain correct matrices with unsymmetric contributions.

Some simulations using matrices can include multiple analysis procedures using symmetric and unsymmetric solvers. For example, some analyses can include the following steps:

- A general static step using the unsymmetric solver.
- A real frequency extraction step. (Real eigensolvers implemented in Abaqus require the stiffness and mass matrices to be symmetric. Even for the special case of solving the unsymmetric coupled acoustic-structural eigenproblem, we assume that the structural and acoustic operator parts are symmetric and that only the coupling terms violate the symmetry.)

To use matrices generated by Abaqus in such simulations, both symmetric and unsymmetric "versions" of the matrices should be generated. You can generate symmetric and unsymmetric matrices from the same finite element model by specifying two separate matrix generation steps and selecting the unsymmetric solver for one of them (for more information, see *Using Matrices*).

Generating Element Matrices

By default, the matrix generation procedure generates global matrices for a model in assembled form. The generated global matrices are assembled from the local element matrices and include contributions from matrix input data. Abaqus/Standard offers an option to generate global matrices in element-by-element form. Instead of global (assembled) matrices, local element matrices are generated. If you choose to generate local element matrices for a model containing matrix input data, Abaqus/Standard calculates and stores only element matrices; the matrix input data are ignored.

You can use the matrix assembly utility (*Element Matrix Assembly Utility*) to assemble element matrices in the SIM document and/or write assembled matrices to text files.

Input File Usage: **MATRIX GENERATE*, ELEMENT BY ELEMENT

Generating Matrices for a Part of the Model

You can generate matrices for a part of the model defined by an element set, the source contributions, or the physical fields.

Defining the Element Set from Which to Generate Matrices

By default, the matrix generation procedure generates matrices for a whole model. Abaqus/Standard can generate matrices for a part of the model defined by an element set.

Input File Usage: **MATRIX GENERATE*, ELSET=*element set name*

Defining the Source Contributions from Which to Generate Matrices

By default, the matrix generation procedure generates matrices from the finite elements and from the matrix input data used in the finite element model. Abaqus/Standard can generate matrices for a part of the model defined by the finite elements only or by the matrix input contributions only.

Input File Usage: Use the following option to generate matrices from the finite elements and from the matrix input data:

**MATRIX GENERATE*, SOURCE=ALL (default)

Use the following option to generate matrices including only contributions from the finite elements:

**MATRIX GENERATE*, SOURCE=ELEMENTS

Use the following option to generate matrices including only contributions from the matrix input data:

**MATRIX GENERATE*, SOURCE=MATRIX INPUT

Defining the Physical Fields from Which to Generate Matrices

You can also define a part of the model for the matrix generation by the physical field. The matrix generation procedure can generate matrices for mechanical and acoustic fields. By default, the matrix generation procedure generates matrices for the structural and acoustic parts of the model. You can generate matrices for only the structural part or for only the acoustic part of the model. Abaqus/Standard does not support generated matrices containing acoustic degrees of freedom in the matrix usage model (see *Using Matrices*).

Input File Usage: Use the following option to generate matrices for the structural and acoustic parts of the model:

**MATRIX GENERATE*, FIELD=ALL (default)

Use the following option to generate matrices for only the structural part of the model:

**MATRIX GENERATE*, FIELD=MECHANICAL

Use the following option to generate matrices for only the acoustic part of the model:

**MATRIX GENERATE*, FIELD=ACOUSTIC

Evaluating Frequency-Dependent Material Properties

When frequency-dependent material properties are specified in the model definition, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in global matrix generation. If you do not choose the frequency, Abaqus/Standard evaluates the matrices at zero frequency and does not consider the contributions from frequency-domain viscoelasticity.

Input File Usage: **MATRIX GENERATE*, PROPERTY EVALUATION=*frequency*

Applying Multipoint Constraints

If multipoint constraints are used in the model, by default, the generated matrices include only entries for the independent degrees of freedom. You can skip applying the multipoint constraints in the generated model.

Input File Usage: Use the following option to generate the matrices with applied multipoint constraints:

**MATRIX GENERATE*, MPC=YES (default)

Use the following option to skip the multipoint constraints during the matrix generation:

**MATRIX GENERATE*, MPC=NO

Specifying Interface Nodes

You can use the matrix generation procedure to specify some of the nodes as "interface nodes." By specifying interface nodes, you can reduce the number of nodes required for including matrices in the matrix usage analysis. For example, you may want to identify only the nodes required for attaching a subcomponent (a matrix-based subassembly) to the matrix usage model.

Input File Usage: **MATRIX GENERATE*, INTERFACE NODES=*node set name*

Checking Generated Matrices

In a matrix generation analysis, you can check the quality of the generated global stiffness and mass matrices. The matrix check generates six "artificial" rigid body modes and projects the matrices onto the rigid body modal subspace. It is expected that the projected 6×6 stiffness matrix (also known as the rigid body energy matrix) is close to zero in the absence of the boundary conditions and constraints. To perform the stiffness matrix quality check, the boundary conditions and multipoint constraints can be suppressed in the matrix generation procedure. The total inertia statistics for the model are extracted from the projected 6×6 rigid body mass matrix. You can specify the center of rotation for creating the artificial rotational rigid body modes and calculating the global inertia tensor.

You can modify the tolerance values to use for the matrix check. If you request the matrix quality check, the check results are printed in the data (`.dat`) file.

You can decide whether problems reported by the matrix check are treated as errors. By default, problems reported by the matrix check are intended as warnings.

Input File Usage: Use the following option to request the matrix check:

**MATRIX CHECK*

Use the following option to treat any problem reported by the matrix check as an error:

**MATRIX CHECK*, ERROR

Use the following option to specify the center of rotation of the coordinate frame to use for the matrix check:

**MATRIX CHECK*, REFERENCE NODE=*node_id*

Use the following option to modify the tolerances to use for the matrix check:

**MATRIX CHECK*, TOLERANCE=ON

Export

In a matrix generation analysis, you can write generated matrices to files in specific formats for selected workflows.

Input File Usage: Use the following option to generate the assembled mass matrix in sparse OUTPUT4 format for aeroelastic simulations with the ZAERO™ software suite from ZONA Technology, Inc.:

**MATRIX EXPORT*, WORKFLOW=ZAERO

Initial Conditions

Matrix generation is a linear perturbation procedure. Therefore, the initial state for the matrix generation step is the state of the model at the end of the last general analysis step. The generated matrix includes initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the analysis.

Boundary Conditions

Boundary conditions can be defined or modified in a matrix generation step. For more information on defining boundary conditions, see *Boundary Conditions*. Any boundary conditions that are defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Loads

All types of loads can be applied in the load cases for a matrix generation step. Real and imaginary parts of the load vectors will be generated for the complex loads. For more information on applying loads, see *About Loads*. Any loads that are defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Predefined Fields

All types of predefined fields can be specified in a matrix generation procedure. For more information on specifying predefined fields, see *Predefined Fields*. Any predefined fields that are defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Material Options

All types of materials that are available in Abaqus/Standard can be used in a matrix generation procedure.

Elements

All types of elements that are available in Abaqus/Standard can be used in the matrix generation procedure.

Generating Matrices for Models Containing Solid Continuum Infinite Elements

Solid continuum infinite elements (CIN-type elements) have different formulations in static and dynamic analyses. Therefore, when generating matrices for a model containing solid continuum infinite elements, you must specify whether to use the static or dynamic formulation.

Input File Usage: Use the following option to select the static formulation for solid infinite elements:

*MATRIX GENERATE, SOLID INFINITE FORMULATION=STATIC

Use the following option to select the dynamic formulation for solid infinite elements:

*MATRIX GENERATE, SOLID INFINITE FORMULATION=DYNAMIC

Output

In a matrix generation analysis, you can output the stiffness, mass, viscous damping, structural damping, and load matrices to text files. Several formats are available for the matrix output, as discussed below. Matrices are copied from the `.sim` file and output in text files that use the following naming conventions:

jobname_matrixN`.mtx`

where *jobname* is the name of the input file or analysis job, *matrix* is a four-letter identifier indicating the matrix type (as outlined in *Table 1*), and *N* is the number associated with the Abaqus analysis step generating the matrices.

Table 1: Identifiers used in the generated matrix file name.

Identifier	Matrix Type
`STIF`	Stiffness matrix
`MASS`	Mass matrix
`DMPV`	Viscous damping matrix
`DMPS`	Structural damping matrix
`LOAD`	Load matrix

For example, if a stiffness matrix generation is performed in the third step of an analysis job named `VehicleFrame`, the matrix is output to a file named `VehicleFrame_STIF3.mtx`.

Alternatively, the matrices can be output into a NASTRAN DMIG-s format file that uses the following naming convention:

jobname`_X`*N*`.bdf`

where *jobname* is the name of the input file or analysis job and *N* is the number associated with the Abaqus analysis step generating the matrices. For example, if the matrix generation is performed in the third step of an analysis job named `VehicleFrame`, the matrices are output to a file named `VehicleFrame_X3.bdf`.

Input File Usage: Use the following option to output the stiffness matrix:

*MATRIX OUTPUT, STIFFNESS

Use the following option to output the mass matrix:

*MATRIX OUTPUT, MASS

Use the following option to output the viscous damping matrix:

*MATRIX OUTPUT, VISCOUS DAMPING

Use the following option to output the structural damping matrix:

*MATRIX OUTPUT, STRUCTURAL DAMPING

Use the following option to output the load matrix:

*MATRIX OUTPUT, LOAD

Matrix Input Text Format

This default text format creates matrix files consistent with the format used by the matrix definition technique in Abaqus/Standard (see *Using Matrices*).

An Abaqus/Standard model may contain user-defined nodes and internal nodes. Abaqus/Standard can internally create nodes associated with internal degrees of freedom (for example, Lagrange multipliers or generalized displacements). This format uses negative numbers as labels of the internal nodes.

Input File Usage: *MATRIX OUTPUT, FORMAT=MATRIX INPUT

Format of the Operator Matrix

The assembled sparse matrix operator data are written to the text file as a series of comma-separated lists. Each row in the file represents a single matrix entry; a row is written as a comma-separated list with the following elements:

1. Row node label
2. Degree of freedom for row node
3. Column node label
4. Degree of freedom for column node
5. Matrix entry

Format of the Load Matrix

Nonzero entries in load matrices, which represent right-hand-side vector data, are written to the text file as a comma-separated list with the following elements:

1. Node label
2. Degree of freedom
3. Right-hand-side vector entry

The format of the load vectors and heading labels is based on the Abaqus keyword interface, which allows the generated loads to be easily applied in other Abaqus analyses. Each load vector uses the following headings to indicate the real and imaginary portions of the load:

```
*CLOAD, REAL
*CLOAD, IMAGINARY
```

If the matrix generation step has multiple load cases, the load matrices for each load case are wrapped by the following labels in the generated text file:

```
*LOAD CASE
…
*END LOAD CASE
```

Including Generated Matrix Data and Generated Loads in Another Abaqus Model

The generated sparse matrix data and generated loads that are output in matrix input text format can be included in another Abaqus model.

Input File Usage: Use the following options to include a matrix:

**MATRIX INPUT*
**INCLUDE*, INPUT=*matrix file*

Use the following options to apply loads in an analysis step:

**STEP*
**INCLUDE*, INPUT=*load file*

Labeling Text Format

This format converts internal Abaqus node labels into large positive numbers that are acceptable for Abaqus matrix input data. This is the only difference between the labeling text format and the default matrix input text format. All nodes of the matrix generated in the labeling text format have positive labels, and Abaqus/Standard analysis treats them as user-defined nodes. That can confuse Abaqus equation solver and result in aborting the analysis or generating incorrect results. You should avoid using this format when it is possible.

Input File Usage: **MATRIX OUTPUT*, FORMAT=LABELS

Coordinate Text Format

You can generate text files in which the matrix is formatted according to the common mathematical coordinate format. This format is commonly used in mathematics programs such as MATLAB®.

Each row in a coordinate-formatted file corresponds to a matrix entry; a row is written as a comma-separated list with the following elements:

1. Row number
2. Column number
3. Matrix entry

For load matrices, which represent right-hand-side vector data, each row in the text file is written with the following elements:

1. Equation (row) number
2. Right-hand-side vector entry

Commented load case options are written to the output file to indicate the load cases.

Input File Usage: **MATRIX OUTPUT*, FORMAT=COORDINATE

DMIG Format

You can output the generated matrices into a NASTRAN DMIG-s format (`.bdf`) file. See NASTRAN documentation for details.

Input File Usage: **MATRIX OUTPUT*, FORMAT=DMIG

User Element Format

You can output the generated matrices into the format required for linear user element matrix input. Both symmetric and unsymmetric matrices can be generated. The file name containing the matrices includes the job name followed by the letter x and the step number from which the matrix was generated:

jobnameXstep number`.UEL`

Input File Usage: **MATRIX OUTPUT*, FORMAT=USER ELEMENT

Outputting Matrices in Element-by-Element Form

If matrices are generated in element-by-element form, you can write them in element-by-element form. When you generate text files using the matrix input or labeling format, each row in the file represents a single matrix entry; a row is written as a comma-separated list with the following elements:

1. Element label
2. Row node label
3. Degree of freedom for row node
4. Column node label
5. Degree of freedom for column node
6. Matrix entry

For load matrices, which represent right-hand-side vector data, each row in the text file is written with the following elements:

1. Element label
2. Node label
3. Degree of freedom
4. Right-hand-side vector entry

Coordinate format is also available for element-by-element global matrix generation. Each row in a coordinate-formatted file corresponds to a matrix entry; a row is written as a comma-separated list with the following elements:

1. Element number
2. Row number
3. Column number
4. Matrix entry

For load matrices each row in the text file is written with the following elements:

1. Element number
2. Equation (row) number
3. Right-hand-side vector entry

Input File Template

```
*HEADING
…
**
*STEP
Options to define the preloading history for the model.
*END STEP
**
*STEP
*MATRIX GENERATE, STIFFNESS, MASS, VISCOUS DAMPING,
STRUCTURAL DAMPING, LOAD
*MATRIX OUTPUT, STIFFNESS, MASS, VISCOUS DAMPING,
STRUCTURAL DAMPING, LOAD, FORMAT=MATRIX INPUT
```

```
*BOUNDARY
Options to define the boundary conditions for the matrix generation step.
**
*LOAD CASE, NAME=LC1
Options to define the loading for the first load case.
*END LOAD CASE
**
*LOAD CASE, NAME=LC2
Options to define the loading for the second load case.
*END LOAD CASE
Any number of load cases can be defined.
*END STEP
```

Generating Matrices from Nonlinear Procedures

Products: Abaqus/Standard

Overview

Matrix generation from nonlinear structural and thermal procedures:

- allows for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the thermal conductivity, heat capacity, and load at specific times;
- writes matrix data to a binary SIM document that can be further processed by Abaqus; and
- can be used as part of a static analysis or an uncoupled heat transfer analysis.

References:

- *Element Matrix Assembly Utility*
- **OPERATOR OUTPUT*

Introduction

A linearized finite element model can be summarized in terms of load vector and left-side matrices that represent mass and stiffness for static analysis or heat capacity and thermal conductivity for uncoupled heat transfer analysis. This mathematical abstraction serves various purposes. For example, you can use these matrices to exchange model data with other users, vendors, or software packages without exchanging mesh or material information. You can also use these matrices in techniques such as model order reduction. This abstraction can also be extended to transient nonlinear problems, which can be treated as a series of piecewise linear models constructed from matrix data at discrete times.

Matrix generation from nonlinear analyses accounts for all current boundary conditions, loads, and material responses in the model. The generated matrices are stored in a SIM document named *jobname*_STATIC*n*.sim and *jobname*_THERM*n*.sim for static and heat transfer analyses, respectively. Here *jobname* is the name of the input file or analysis job, and *n* is the number of the Abaqus static or heat transfer step that generates the matrices.

Defining Matrix Types for a Static Analysis

The discretization in space of the static equation (see *About procedures and basic equations*) is

$$\int_{V^0} \boldsymbol{\beta}_N : \boldsymbol{\tau}^c \, dV^0 = \int_S \mathbf{N}_N^T \cdot \mathbf{t} \, dS + \int_V \mathbf{N}_N^T \cdot \mathbf{f} \, dV,$$

where

$\boldsymbol{\beta}_N$ defines the strain variation from the variations of the kinematic variables,

$\boldsymbol{\tau}^c$ is the Kirchhoff stress,

$\mathbf{N}_N$ are the finite element interpolation functions,

$\mathbf{t}$ is the traction per unit current area,

$\mathbf{f}$ is the prescribed force per unit volume,

V^0 is the reference volume of the domain,

V is the current volume of the domain, and

S is the surface bounding this volume.

The stiffness matrix, $\mathbf{K}$, is defined as

$$K_{MN} = \int_{V^0} \boldsymbol{\beta}_M : \mathbf{H} : \boldsymbol{\beta}_N \, dV^0 + \int_{V^0} \boldsymbol{\tau}^c : \partial_N \boldsymbol{\beta}_M \, dV^0 - \int_S \mathbf{N}_M^T \cdot \mathbf{Q}_N^S \, dS - \int_V \mathbf{N}_M^T \cdot \mathbf{Q}_N^V \, dV,$$

where $\mathbf{H}$ is defined in terms of the current state, direction of straining, etc. and on the kinematic assumptions used to form the generalized strains. See *Mechanical Constitutive Theories* for a detailed discussion of forming $\mathbf{H}$ for the material models available in Abaqus. $\mathbf{Q}_N^S$ and $\mathbf{Q}_N^V$ depend on the type of load being considered and represent the variation of the load vector with nodal variables on the surface S and the volume V, respectively.

For special workflows, the mass matrix $\mathbf{M}$ (see *Implicit dynamic analysis*) is defined as

$$M_{NM} = \int_{V^0} \rho^0 \mathbf{N}_N \cdot \mathbf{N}_M dV^0,$$

where ρ^0 is the reference density.

Defining Matrix Types for an Uncoupled Heat Transfer Analysis

The continuous time description of the spatially discretized heat transfer equation (see *Uncoupled heat transfer analysis*) is

$$\begin{aligned}\int_V N^N \rho \dot{U} dV = & -\int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV \\ & + \int_V N^N r \, dV + \int_{S_q} N^N q \, dS,\end{aligned}$$

where θ is the temperature field, N^N are the finite element interpolation functions, ρ is the material density, $\dot{U}$ is the material time derivative of the internal energy, $\mathbf{k}$ is the (possibly anisotropic) conductivity matrix, r is the prescribed heat flux per unit volume, V is the volume of the domain, and S_q is the surface on which heat flux per unit area q is either directly prescribed or specified through film and radiation conditions.

The external flux vector $\mathbf{P}$ is defined as

$$P^N = \int_V N^N r \, dV + \int_{S_q} N^N q \, dS.$$

The internal flux vector $\mathbf{I}$ is defined as

$$I^N = -\int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV.$$

The net flux vector $\mathbf{F}$ is defined as the sum of the internal flux vector $\mathbf{I}$ and the external flux vector $\mathbf{P}$. The heat capacity matrix $\mathbf{C}$ is defined as

$$C^{NM} = \int_V N^N \rho \frac{\partial U}{\partial \theta} N^M dV.$$

The thermal conductivity matrix $\mathbf{K}$ is defined as

$$K^{NM} = -\frac{\partial}{\partial \theta^M}\left[-\int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV + \int_V N^N r\, dV + \int_{S_q} N^N q\, dS\right].$$

That is, the thermal conductivity matrix is the negated derivative of the net flux vector with respect to the nodal temperature vector $\boldsymbol{\theta}$ and, hence, includes the effect of temperature-dependent flux conditions such as film and radiation.

Generating Assembled or Element-by-Element Matrices

Matrices are written to the SIM document in assembled or element-by-element forms.

Input File Usage: Use the following option to write matrices in assembled form:

**OPERATOR OUTPUT*, OPERATOR=*ASSEMBLED*

Use the following option to write matrices in element-by-element form:

**OPERATOR OUTPUT*, OPERATOR=*ELEMENT BY ELEMENT*

Specifying the Matrix Type

You can generate matrices representing the following model features:

- Stiffness
- Mass
- Heat capacity
- Thermal conductivity
- Loads
- Solutions
- Free body response

The thermal conductivity matrix has an unsymmetric contribution if the thermal conductivity property is temperature dependent. This term is taken into account only if the unsymmetric solver has been activated in the step definition (see *Defining an Analysis*).

The load matrix in uncoupled heat transfer analysis contains either the nodal external flux vector or the net flux vector corresponding to the loading defined in the heat transfer step.

Input File Usage: Use the following option to generate the stiffness matrix from a static analysis:

**OPERATOR OUTPUT*, STIFFNESS

Use the following option to generate the mass matrix from a static analysis:

**OPERATOR OUTPUT*, MASS

Use the following option to generate the load vector from a static analysis:

**OPERATOR OUTPUT*, LOAD

Use the following option to generate the free body response matrix from a static analysis:

**OPERATOR OUTPUT*, FREE BODY RESPONSE

Use the following option to generate the solution vector from a static analysis:

**OPERATOR OUTPUT*, SOLUTION

Use the following option to generate the heat capacity matrix from an uncoupled heat transfer analysis:

**OPERATOR OUTPUT*, HEAT CAPACITY

Use the following option to generate the thermal conductivity matrix from an uncoupled heat transfer analysis:

**OPERATOR OUTPUT*, THERMAL CONDUCTIVITY

Use the following option to generate the external flux vector from an uncoupled heat transfer analysis:

**OPERATOR OUTPUT*, LOAD=*EXTERNAL*

Use the following option to generate the net flux vector from an uncoupled heat transfer analysis:

**OPERATOR OUTPUT*, LOAD=*NET*

Generating Matrices for a Part of the Model

By default, matrices are generated for all supported element types in the model. You can request that Abaqus/Standard generate matrices for a part of the model defined by an element set.

Input File Usage: **OPERATOR OUTPUT*, ELSET=*element set name*

Specifying the Frequency of Matrix Generation

By default, matrices are generated for every increment in the step in which it is requested. You can request that Abaqus/Standard generate matrices at a specified frequency.

Input File Usage: Use one of the following mutually exclusive options:

**OPERATOR OUTPUT*, FREQUENCY=*output frequency*

**OPERATOR OUTPUT*, NUMBER INTERVAL=*number of intervals*

Initial Conditions

Matrix generation occurs in a general analysis procedure. Therefore, the generated matrices include the effect of initial conditions when applicable.

Boundary Conditions

Prescribed boundary conditions are either imposed or not imposed on the generated matrices and load vectors, based on user-defined requests.

Loads

All types of loads supported in static analysis and uncoupled heat transfer analysis can be used for matrix generation. For more information on applying loads, see *About Loads*. For uncoupled heat transfer analysis, load types that are functions of temperature (such as film and radiation) contribute additional “load stiffness” terms to the thermal conductivity matrix.

Predefined Fields

All types of predefined fields can be specified for matrix generation. For more information on specifying predefined fields, see *Predefined Fields*.

Material Options

All types of materials that are available in Abaqus/Standard for static and uncoupled heat transfer analyses can be used for matrix generation.

Elements

All element types are supported in matrix generation for static analysis.

Only continuum diffusive heat transfer elements and thermal contact elements are supported for thermal matrix generation. Thermal matrices are written to the SIM document only for supported elements.

Output

The generated matrices are written to the output SIM document in either element-by-element or assembled form. For efficiency, only nonzero matrix entries are stored in the SIM document. If the matrix is symmetric, only the nonzero entries in the upper triangular portion of the matrix are stored. You can use the matrix assembly utility (*Element Matrix Assembly Utility*) to assemble element matrices in the SIM document and/or write assembled matrices to text files.

Limitations

Constraints that are implemented using the degree-of-freedom elimination technique (such as tie constraints) are not processed for thermal matrix output. In addition, cavity radiation effects are not considered for thermal matrix output.

Input File Template

The following template can be used to generate matrices for an Abaqus/Standard static analysis:

```
*HEADING
…
**
*STEP
Options to define a static analysis.
…
*BOUNDARY
Options to define the boundary conditions for the static step.
**
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to define loads
**
*OPERATOR OUTPUT, OPERATOR=ASSEMBLED, MASS, STIFFNESS, LOAD,
FREE BODY RESPONSE, SOLUTION, FREQUENCY=1
**
```

```
Options to define the output requests for the static step.
**
*END STEP
```

The following template can be used to generate matrices for an Abaqus/Standard uncoupled heat transfer analysis:

```
*HEADING
…
**
*STEP
Options to define an uncoupled heat transfer analysis.
…
*BOUNDARY
Options to define the boundary conditions for the heat transfer step.
**
*CFLUX and/or *DFLUX and/or *DSFLUX
Data lines to define thermal loading
*FILM and/or *SFILM and/or *RADIATE and/or *SRADIATE
Data lines to define convective film and radiation conditions
**
*OPERATOR OUTPUT, OPERATOR=ASSEMBLED, HEAT CAPACITY, THERMAL CONDUCTIVITY,

LOAD=NET, FREQUENCY=1
**
Options to define the output requests for the heat transfer step.
**
*END STEP
```

Symmetric Model Generation, Results Transfer, and Analysis of Cyclic Symmetry Models

In this section:

- *Symmetric Model Generation*
- *Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*
- *Analysis of Models that Exhibit Cyclic Symmetry*

Symmetric Model Generation

Products: Abaqus/Standard

References:

- **SYMMETRIC MODEL GENERATION*

Overview

A three-dimensional model can be created in Abaqus/Standard by:

- revolving an axisymmetric model about its axis of revolution;
- revolving a single three-dimensional sector about its axis of symmetry; or
- combining two parts of a symmetric three-dimensional model, where one of the parts is the original model and the other part is obtained by reflecting the original model through either a symmetry line or a symmetry plane.

Abaqus/Standard also provides for the transfer of the solution obtained in the original analysis onto the new model (see *Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*).

Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be used to generate a new model.

Model Generation

The symmetric model generation capability can be used to create a three-dimensional model by revolving an axisymmetric model about its axis of revolution, by revolving a single three-dimensional sector about its axis of symmetry, or by combining two parts of a symmetric model, where one part is the original model and the other part is the original model reflected through a line or a plane. The original model must have been saved to a restart file. The symmetric model generation capability is not available for models defined in terms of an assembly of part instances. Therefore, an element set name or a node set name containing quotation marks is not supported.

An entire three-dimensional model—including nodes, elements, section definitions, material and orientation definitions, rebar, and contact pair definitions—is generated from the original model. Symmetric model generation from a model with general contact is not allowed. You must redefine most types of kinematic constraints (*About Kinematic Constraints*). However, surface-based constraints (*Mesh Tie Constraints*) and embedded element constraints (*Embedded Elements*) defined in the original model will be generated automatically in the new three-dimensional model. Changes made to the model as part of the history data—element or contact pair removal/reactivation (*Element and Contact Pair Removal and Reactivation*) or changes to friction properties (*Changing Friction Properties during an Abaqus/Standard Analysis*)—will not be transferred to the new model. Such changes will have to be redefined in the history data of the new model. All element and node sets defined in the original model will be used in the new model. These sets will contain all of the new elements and nodes that originated from the original sets.

Additional nodes, elements, contact surfaces, etc. can also be defined to create parts of the model that were not specified in the original model. You must ensure that the numbering of these nodes and elements does not conflict with those used by the symmetric model generation capability. You can control the node and element numbering in the new model (as described below for each type of revolved model) so that you can define additional parts of the model without the risk of conflicting element and node labels. The smallest node/element number used

in defining additional parts of the new model should be greater than the largest node/element number generated by the symmetric model generation capability.

Eliminating Duplicate Nodes

Duplicate nodes will be generated in certain situations. Such nodes can be eliminated to ensure that the mesh is connected properly. Duplicate nodes can be generated on the axis of revolution of a revolved model, on the connection planes between sectors of a periodic model, and on the connection plane between the two parts of a reflected model. You can specify the tolerance distance, *d*, to be used in the search for duplicate nodes. The default distance is 1.0% of the average element dimension. In some cases a tolerance distance that is smaller than the default value needs to be specified if, for example, the distance between two nodes on one of the connection planes in the original sector of a periodic model is smaller than the default tolerance distance. Closely spaced nodes elsewhere in the model, such as between interface surfaces or on parts of the model that are generated with any of the other model definition options, will not be eliminated.

Input File Usage: Use one of the following options to specify the tolerance to be used in the search for duplicate nodes:

**SYMMETRIC MODEL GENERATION*, PERIODIC, TOLERANCE=*d*
**SYMMETRIC MODEL GENERATION*, REVOLVE, TOLERANCE=*d*
**SYMMETRIC MODEL GENERATION*, REFLECT, TOLERANCE=*d*

Writing the New Model Definition to an External File

You can specify the name of an external file (without an extension) to which the data for the new model definition will be written. The extension `.axi` will be added to the file name provided. The file can be edited to modify or to extend the model generated by Abaqus/Standard.

Input File Usage: Use one of the following options:

**SYMMETRIC MODEL GENERATION*, PERIODIC, FILE NAME=*name*
**SYMMETRIC MODEL GENERATION*, REVOLVE, FILE NAME=*name*
**SYMMETRIC MODEL GENERATION*, REFLECT, FILE NAME=*name*

Identifying the Restart Files

The symmetric model generation capability uses the restart (`.res`), analysis database (`.stt` and `.mdl`), and part (`.prt`) files from the old model to generate the new model. The name of the restart files from the old model must be specified when the new analysis is executed by using the **oldjob** parameter in the command for running Abaqus or by answering a request made by the command procedure (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Verifying the New Model

It is recommended that you verify the new model carefully before an analysis is performed. The symmetric model generation capability requires only information stored in the restart file during a data check run to generate the new model, which allows you to verify the new model before the analysis of the original model is performed. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Revolving an Axisymmetric Cross-Section

You can create a three-dimensional model by revolving the cross-section of a two-dimensional axisymmetric model about a symmetry axis starting at a prescribed reference plane, $\theta = 0.0°$. Both the symmetry axis and

reference plane of the new three-dimensional model can be oriented in any direction with respect to the global coordinate system (see *Figure 1*). A nonuniform discretization in the circumferential direction can be specified.

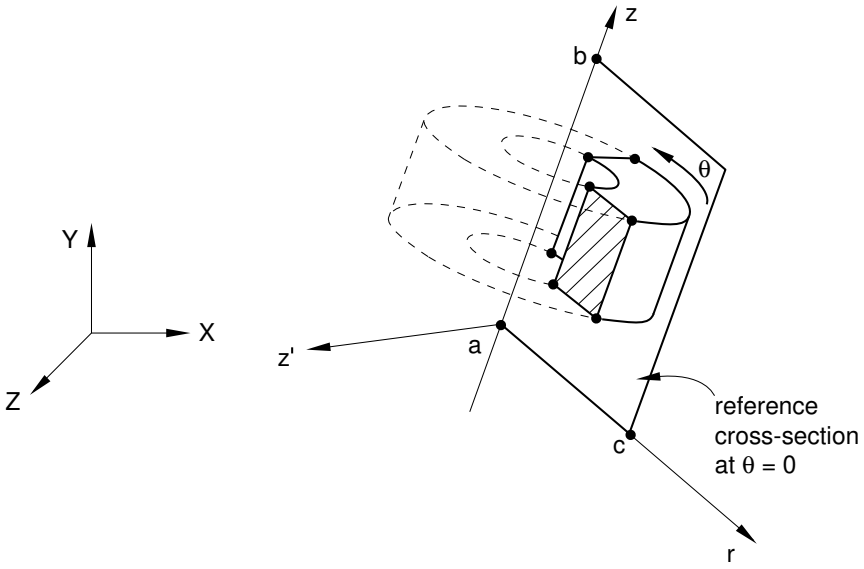


Figure 1: Revolving an axisymmetric cross-section.

Specify the coordinates of points *a*, *b*, and *c* shown in *Figure 1*, followed by a list that defines the discretization in the circumferential direction containing the segment angle, number of elements per segment, and the bias ratio of the segment. Several segment angles, each with a different number of element subdivisions and a different bias ratio, can be used to define the complete discretization around the circumference of the revolved model. The endpoint of a cross-section revolved through 360.0° will always be connected to the origin of revolution, $\theta = 0.0°$, regardless of the value specified for the duplicate node tolerance.

Input File Usage: **SYMMETRIC MODEL GENERATION*, REVOLVE

Local Orientation System

A local cylindrical orientation system is always used for element output of stress, strain, etc. A default local orientation definition is provided if the material in the original axisymmetric model does not contain an orientation definition. This default orientation is defined with the polar axis of the system along the axis of revolution, with an additional 90.0° rotation about the local 1-direction so that the local axes are 1=radial, 2=axial, and 3=circumferential. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. This orientation system is always provided, even if an orientation is specified in the original axisymmetric model. However, if the results of the axisymmetric analysis are mapped onto the new three-dimensional model (see *Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh*) and an orientation definition is associated with the material in the original model, the original orientation revolved about the axis of symmetry replaces this default orientation definition.

Controlling the New Node and Element Numbering

You can define the increments in numbers between each node and element around the circumference of the three-dimensional model. The numbering starts at the reference cross-section $\theta = 0.0°$. The reference cross-section uses the same numbering as the original axisymmetric model. The defaults are the largest node and element numbers used in the original axisymmetric model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels. Each offset value should be greater

than or equal to the maximum node or element label, respectively, used in the original model. When specifying the offset value, care must be taken that the generated node or element does not exceed the maximum value allowed, which is 999,999,999.

Input File Usage: **SYMMETRIC MODEL GENERATION*, REVOLVE, NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Correspondence between Axisymmetric and Three-Dimensional Elements

The element type used in the original two-dimensional model determines the element type in the new three-dimensional model. You can specify whether the new element should be either a general three-dimensional element or a cylindrical element. General and cylindrical elements can be used in the same model.

Regular axisymmetric elements (CAX), axisymmetric elements with twist (CGAX), shell elements, membrane elements, rigid elements, and surface elements can be used in the two-dimensional model; however, nonlinear axisymmetric elements (CAXA) cannot be used. A two-dimensional model that contains incompatible mode elements; first-order, reduced-integration, continuum elements; shell elements; or rigid elements cannot be used to generate cylindrical elements. The correspondence between the axisymmetric element type and the equivalent three-dimensional element type (general or cylindrical) is shown in *Table 1*.

Table 1: Correspondence between axisymmetric and three-dimensional (general and cylindrical) element types.

Axisymmetric element	General three-dimensional element	Cylindrical element
ACAX3	AC3D6	
CAX3	C3D6	CCL9
CAX3H	C3D6H	CCL9H
CGAX3	C3D6	CCL9
CGAX3H	C3D6H	CCL9H
CGAX3T	C3D6T	
DCAX3	DC3D6	
ACAX4	AC3D8	
CAX4	C3D8	CCL12
CAX4H	C3D8H	CCL12H
CAX4I	C3D8I	
CAX4R	C3D8R	
CAX4RH	C3D8RH	
CGAX4	C3D8	CCL12
CGAX4H	C3D8H	CCL12H
CGAX4R	C3D8R	
CGAX4RH	C3D8RH	
CAX4T	C3D8T	
CAX4RT	C3D8RT	
CAX4HT	C3D8HT	
CAX4RHT	C3D8RHT	
CGAX4T	C3D8T	

Axisymmetric element	General three-dimensional element	Cylindrical element
CGAX4RT	C3D8RT	
CGAX4HT	C3D8HT	
CGAX4RHT	C3D8RHT	
DCAX4	DC3D8	
DCCAX4	DCC3D8	
DCCAX4D	DCC3D8D	
ACAX6	AC3D15	
CAX6	C3D15	CCL18
CAX6H	C3D15H	CCL18H
CGAX6	C3D15	CCL18
CGAX6H	C3D15H	CCL18H
DCAX6	DC3D15	
ACAX8	AC3D20	
CAX8	C3D20	CCL24
CAX8H	C3D20H	CCL24H
CAX8R	C3D20R	CCL24R
CAX8RH	C3D20RH	CCL24RH
CGAX8	C3D20	CCL24
CGAX8H	C3D20H	CCL24H
CGAX8R	C3D20R	CCL24R
CGAX8RH	C3D20RH	CCL24RH
CAX8T	C3D20T	
CAX8RT	C3D20RT	
CAX8HT	C3D20HT	
CAX8RHT	C3D20RHT	
CGAX8T	C3D20T	
CGAX8RT	C3D20RT	
CGAX8HT	C3D20HT	
CGAX8RHT	C3D20RHT	
DCAX8	DC3D20	
SAX1	S4R	
DSAX1	DS4	
SAX2	S8R	
DSAX2	DS8	
MAX1	M3D4R	MCL6
MGAX1	M3D4R	MCL6
MAX2	M3D8R	MCL9

Axisymmetric element	General three-dimensional element	Cylindrical element
MGAX2	M3D8R	MCL9
RAX2	R3D4	
SFMAX1	SFM3D4R	SFMCL6
SFMGAX1	SFM3D4R	SFMCL6
SFMAX2	SFM3D8R	SFMCL9
SFMGAX2	SFM3D8R	SFMCL9

Input File Usage: **SYMMETRIC MODEL GENERATION*, REVOLVE
coordinates of points a and b
coordinates of point c
segment angle, number of elements per segment, bias ratio, CYLINDRICAL or GENERAL

For example, the following input specifies 4 cylindrical elements in a 300° segment and 10 general elements in a 60° segment:

**SYMMETRIC MODEL GENERATION*, REVOLVE
a_x, a_y, a_z, b_x, b_y, b_z
c_x, c_y, c_z
300.0, 4, 1.0, CYLINDRICAL
60.0, 10, 1.0, GENERAL

Limitations

- First- and second-order elements cannot be used together in the axisymmetric model.
- Nonaxisymmetric elements such as springs, dashpots, beams, and trusses will be ignored in the model generation.
- Only surface-based contact pairs can be revolved. Models using general contact cannot be revolved. Contact conditions modeled with contact elements will be ignored in the model generation.
- A two-dimensional model that includes incompatible mode elements; first-order, reduced-integration, continuum elements; shell elements; or rigid elements cannot be used to generate cylindrical elements.
- Rebar with nonuniform spacing in the radial direction of an axisymmetric element cannot be revolved.
- Most types of kinematic constraints cannot be revolved. However, surface-based constraints (*Mesh Tie Constraints*) and embedded element constraints (*Embedded Elements*) defined in the original model will be generated automatically in the new three-dimensional model.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be revolved.
- Spatially varying fields defined using distributions (*Distribution Definition*) cannot be revolved.

Revolving a Three-Dimensional Sector to Create a Periodic Model

You can create a three-dimensional periodic model by revolving a single three-dimensional sector about a symmetry axis. Each generated sector in the periodic model can span the same angle in the circumferential direction, such as in a vented disc, or alternatively, can have a variable angle, such as in a treaded tire. In both cases, each sector always has the same geometry and mesh. Both the symmetry axis and the original three-dimensional sector can be oriented in any direction with respect to the global coordinate system (see *Figure 2*). Mismatched surface meshes can be used between sectors. Both open (the structure has end edges) or closed

loop periodic structures can be generated. If a closed loop periodic structure needs to be created, the sum of the segment angles over all the sectors must be equal to 360°.

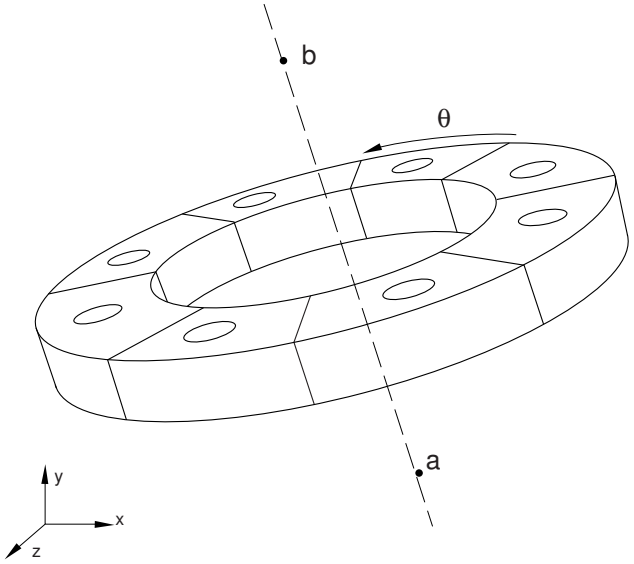


Figure 2: Revolving a three-dimensional sector to form a periodic model.

Defining a Periodic Model with a Constant Angle

To define a periodic model with a constant angle, you must specify the coordinates of points *a* and *b* shown in *Figure 2* to define the symmetry axis. You then define the segment angle, θ (in degrees), of the original sector and the number of three-dimensional repetitive sectors, *N*, including the original sector, in the generated periodic model.

Input File Usage:
**SYMMETRIC MODEL GENERATION*, PERIODIC=CONSTANT
coordinates of points a and b
, N

Defining a Periodic Model with a Variable Angle

To define a periodic model with a variable angle, the surfaces on both sides of the original sector must be completely planar. You specify the coordinates of points *a* and *b* shown in *Figure 2* to define the symmetry axis. You then define the segment angle, θ (in degrees), of the original sector and the number of three-dimensional repetitive sectors, *N*, including the original sector, in the generated periodic model. Next, you specify an additional number of three-dimensional sectors to be generated, *M*, and the angular scaling factor, *f*, in the circumferential direction with respect to the original sector to be applied to these additional sectors. You can define pairs of additional sectors and scaling factors as needed.

Input File Usage:
**SYMMETRIC MODEL GENERATION*, PERIODIC=VARIABLE
coordinates of points a and b
, N
M_1, f_1
M_2, f_2
Etc.

For example, the following input creates a 210° three-dimensional model with 7 sectors with the angles of 20°, 20°, 30°, 30°, 30°, 40°, and 40°, respectively:

*SYMMETRIC MODEL GENERATION, PERIODIC=VARIABLE
a_x, a_y, a_z, b_x, b_y, b_z
20.0,2
3,1.5
2,2.0

Applying Constraints to Symmetric Surfaces with Mismatched Meshes

If the symmetric surfaces in the original sector have precisely matched meshes, as shown in *Figure 3*, any duplicate nodes that are generated will be eliminated automatically to ensure that the mesh is connected properly between the neighboring sectors when revolving the original sector about the symmetry axis to create a periodic model.

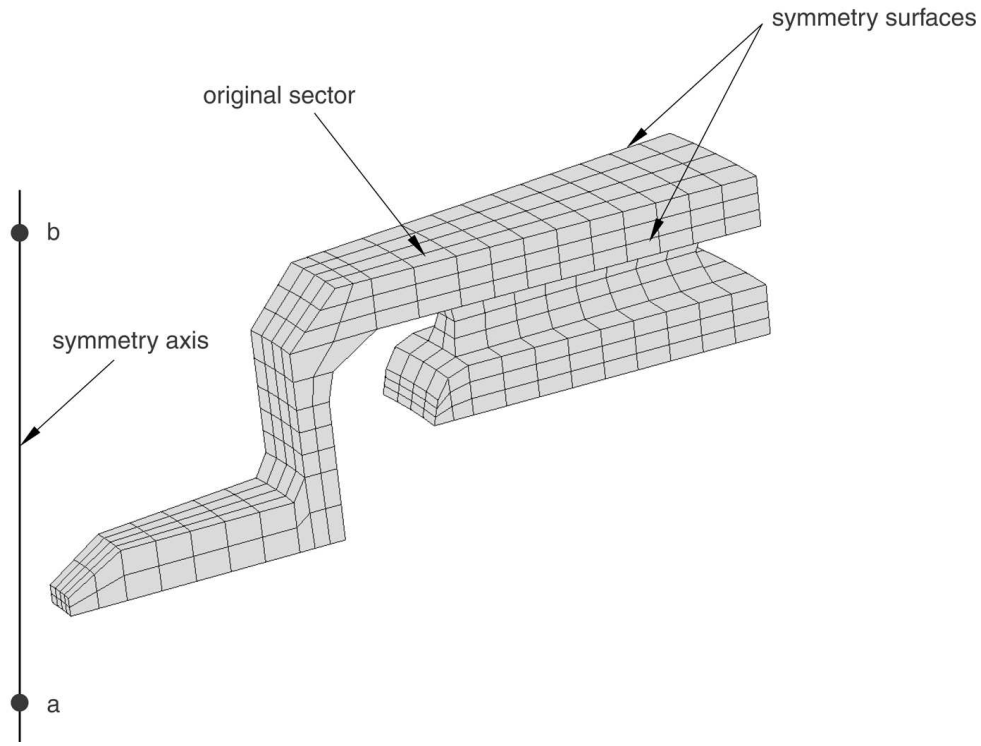


Figure 3: Surfaces with precisely matching meshes on the original sector.

In all other cases you must define one or more pairs of corresponding surfaces on each side of the original sector (see *About Surfaces*) in the original model and specify the pairs of corresponding surfaces in the symmetric model generation definition.

Optionally, you can also specify the tolerance distance within which nodes on one surface of a sector must lie from the corresponding surface of the neighboring sector to be constrained. Nodes on the surface of the sector that are further away from the corresponding surface of the neighboring sector than this distance are not constrained. The default value for the tolerance distance is 5% or 10% of the typical element size in the surfaces of the original sector, depending on whether node-to-surface or surface-to-surface type constraints are used, respectively.

You can also specify whether surface-to-surface (default) or node-to-surface constraints should be used. Constraints between the automatically generated neighboring pairs of corresponding surfaces are then applied with an automatically generated surface-based tie constraint (*Mesh Tie Constraints*) when revolving the original sector about the symmetry axis to create a periodic model. The first surface of each specified pair is the secondary

surface, and all degrees of freedom of the nodes in the surface will be eliminated by internally generated multi-point constraints.

Input File Usage: Use the following options in the original model:

**SURFACE*, NAME=*main*
**SURFACE*, NAME=*secondary*

Use the following option in the new model with a constant angle for each sector:

**SYMMETRIC MODEL GENERATION*, PERIODIC=CONSTANT
a_x, a_y, a_z, b_x, b_y, b_z
, N
secondary, main, tolerance distance, SURFACE or NODE

Use the following option in the new model with a variable angle for each sector:

**SYMMETRIC MODEL GENERATION*, PERIODIC=VARIABLE
a_x, a_y, a_z, b_x, b_y, b_z
, N
M, f
secondary, main, tolerance distance, SURFACE or NODE

Local Orientation System

A local cylindrical orientation system is always used for element output of stress, strain, etc. If an orientation is specified in the original three-dimensional sector (see *Orientations*), the orientation system in the new model is defined by revolving the original orientation system about the symmetry axis. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. If the material in the original three-dimensional sector does not contain an orientation definition, a default local orientation definition is provided. This default orientation is defined by revolving the global coordinate system in the original model about the axis of symmetry in the new model.

Controlling the New Node and Element Numbering

You can define the increments in numbers between each node and element around the circumference of the three-dimensional model. The numbering starts at the original three-dimensional repetitive sector. The original three-dimensional repetitive sector uses the same numbering as the original model. The defaults are the largest node and element numbers used in the original model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels. Each offset value should be greater than or equal to the maximum node or element label, respectively, used in the original model. When specifying the offset value, care must be taken that the generated node or element does not exceed the maximum value allowed, which is 999,999,999.

Input File Usage: **SYMMETRIC MODEL GENERATION*, PERIODIC,
NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Limitations

- Only surface-based contact pairs can be revolved. Models using general contact cannot be revolved. Contact conditions modeled with contact elements will be ignored in the model generation.
- Most types of kinematic constraints cannot be revolved. However, surface-based constraints (*Mesh Tie Constraints*) and embedded element constraints (*Embedded Elements*) defined in the original model will be

generated automatically in the new three-dimensional model. One exception is that surface-based ties for enforcing cyclic symmetric constraints are not revolved.

- Surface-based distributed coupling constraints—e.g., couplings (*Coupling Constraints*), shell-to-solid couplings (*Shell-to-Solid Coupling*), and fasteners (*Mesh-Independent Fasteners*)—cannot be revolved and must be redefined.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be revolved. Beam and frame elements cannot be revolved.
- Spatially varying fields defined using distributions (*Distribution Definition*) cannot be revolved.

Reflecting a Partial Three-Dimensional Model

You can create a three-dimensional model by combining two parts of a symmetric three-dimensional model. One of the parts is the original model, and the other part is obtained by reflecting the original model through a symmetry line (*Figure 4*) or plane (*Figure 5*).

Specify the coordinates of points *a*, *b*, and (if required) *c* shown in *Figure 4* and *Figure 5*.

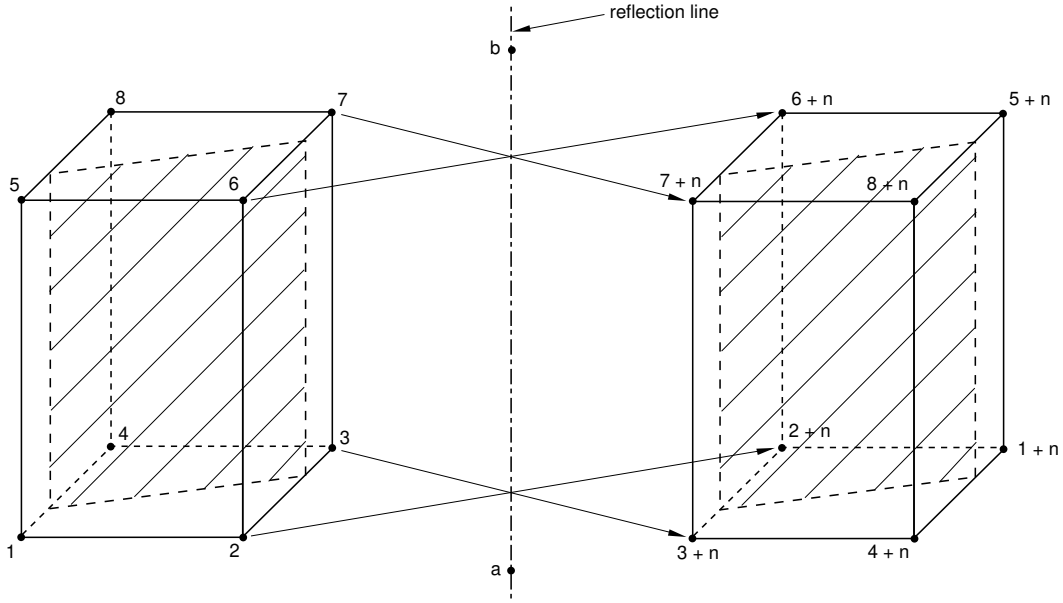


Figure 4: Reflecting a three-dimensional model through line *ab* with node offset *n*.

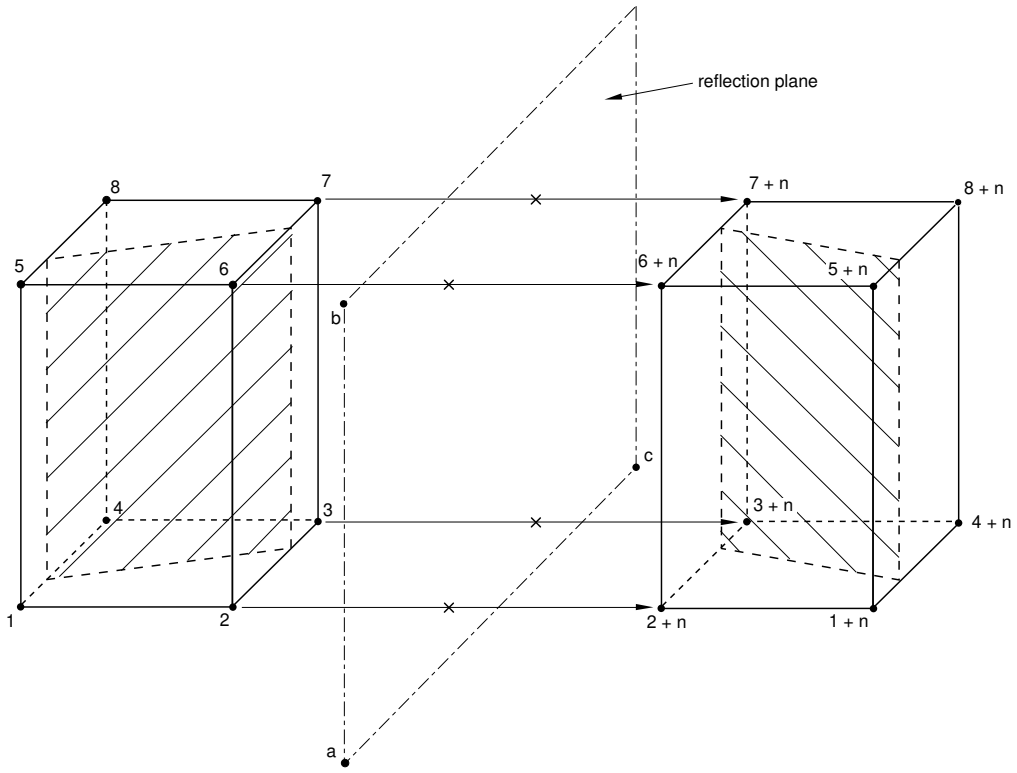


Figure 5: Reflecting a three-dimensional model through a plane *abc* with node offset *n*.

Input File Usage: Use one of the following options:

**SYMMETRIC MODEL GENERATION*, REFLECT=LINE
**SYMMETRIC MODEL GENERATION*, REFLECT=PLANE

Controlling the New Node and Element Numbering

You can specify constants that must be added to the original node and element numbers for numbering the reflected part of the three-dimensional model. The defaults are the maximum node and element numbers used in the original model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels.

Input File Usage: **SYMMETRIC MODEL GENERATION*, REFLECT,
NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Limitations

- Only surface-based contact pairs can be reflected. Models using general contact cannot be reflected. Contact conditions modeled with contact elements will be ignored in the model generation.
- You must ensure that main surfaces remain continuous after reflection. A discontinuous surface is created when the surface in the original model does not intersect the connection plane between the two parts of the symmetric structure.
- Rigid surfaces cannot be reflected. The rigid surface definition of the original model is simply repeated in the new model. You must, therefore, specify the complete rigid surface in the original model.

- Most types of kinematic constraints cannot be reflected. However, surface-based constraints (*Mesh Tie Constraints*) and embedded element constraints (*Embedded Elements*) defined in the original model will be generated automatically in the new three-dimensional model.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be reflected.
- Nonaxisymmetric elements such as springs, dashpots, beams, and trusses cannot be reflected.
- Spatially varying fields defined using distributions (*Distribution Definition*) cannot be reflected.

Transferring Results from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh

Products: Abaqus/Standard

References:

- *Symmetric Model Generation*
- **SYMMETRIC RESULTS TRANSFER*

Overview

Symmetric results transfer:

- reduces the analysis cost of structures that may first undergo symmetric deformation followed by nonsymmetric deformation later during the loading history;
- can be used to transfer the solution of an axisymmetric model to a three-dimensional model;
- can be used to transfer the solution of the symmetric part of a three-dimensional model to a full three-dimensional model;
- must be used in conjunction with the symmetric model generation capability (see *Symmetric Model Generation*); and
- can be used only to transfer the solution of a stress/displacement, heat transfer, coupled temperature-displacement, or coupled acoustic-structural analysis to a new model.

Transferring the Solution from a Symmetric Mesh or a Partial Three-Dimensional Mesh to a Full Three-Dimensional Mesh

The symmetric results transfer capability can be used to transfer the solution of an axisymmetric model to a three-dimensional model or to transfer the solution of the symmetric part of a three-dimensional model to a full three-dimensional model. The symmetric model generation capability described in *Symmetric Model Generation* must be used to generate the three-dimensional model.

The symmetric results transfer capability is not available for models defined in terms of an assembly of part instances.

The solution that is transferred to the new model consists of the deformed configuration and corresponding material state, which includes strains and all state variables. The nodes are imported with their original coordinates. This solution becomes the initial or base state in the new analysis. Due to different underlying kinematics in axisymmetric and three-dimensional models, the transferred three-dimensional solution may differ slightly from the reference axisymmetric solution.

Specifying the Time at Which the Solution Obtained in the Original Model Must Be Read

You specify the time at which the solution obtained in the original model must be read. The required step and increment or iteration must have been written to the restart files during the original analysis.

Input File Usage: Use the following option if the solution is transferred from any analysis other than a direct cyclic procedure:

**SYMMETRIC RESULTS TRANSFER*, STEP=*step*, INC=*increment*

Use the following option if the solution is transferred from a previous direct cyclic analysis:

**SYMMETRIC RESULTS TRANSFER*, STEP=*step*, ITERATION=*iteration*

Obtaining Equilibrium

You must ensure that the model is in equilibrium at the beginning of the analysis. It is recommended that an initial step definition be included using boundary conditions and loading that match the state of the model from which the results are transferred. An initial time increment equal to the total time should be used for this step to allow Abaqus/Standard to try and achieve the equilibrium in one increment. If needed, Abaqus/Standard can resolve the stress unbalance linearly over the step such that more than one increment is used. You can choose to have the stress unbalance resolved in the first increment of the step instead.

Input File Usage: Use the following option to have Abaqus/Standard resolve the stress unbalance linearly over the step:

**SYMMETRIC RESULTS TRANSFER*, UNBALANCED STRESS=RAMP

Use the following option to have Abaqus/Standard resolve the stress unbalance in the first increment of the step:

**SYMMETRIC RESULTS TRANSFER*, UNBALANCED STRESS=STEP

Identifying the Restart Files

The symmetric results transfer capability uses the restart (`.res`), analysis database (`.stt` and `.mdl`), part (`.prt`), and output database (`.odb`) files from the old analysis to transfer the solution data to the new mesh. The name of the restart files from the old analysis must be specified when the new analysis is executed by using the **oldjob** parameter in the command for running Abaqus or by answering a request made by the command procedure (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Verifying the New Model

It is recommended that you verify that the new model is generated correctly before results are transferred or any analysis is performed. The model generation capability requires only information stored in the restart files during a data check run to generate the new model, which allows you to verify the new model before the analysis of the original model is performed. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus (see *Abaqus/Standard and Abaqus/Explicit Execution*).

Once the model has been verified, the analysis of the original model can be performed and the results can be transferred to the new model.

The transferred solution can be written to the results files by requesting output at the beginning of a step (the zero increment; see *About Output*). This solution can also be viewed in Abaqus/CAE.

Orientation System

When results are transferred from an axisymmetric model to a three-dimensional model, a local cylindrical orientation system is used for element output of stress, strain, etc. A default local orientation definition (*Orientations*) is provided if the material in the original axisymmetric model does not contain an orientation definition. This default orientation is defined with the polar axis of the system along the axis of revolution with an additional 90° rotation about the local 1-direction so that the local axes are 1=radial, 2=axial, and 3=circumferential. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. It is assumed that the material properties

are specified in this system. If, on the other hand, an orientation definition is associated with the material in the original model, the orientation in the new three-dimensional model will be that orientation definition revolved about the axis of symmetry.

When results are transferred from a partial three-dimensional model to a full three-dimensional model by reflecting the partial three-dimensional model, a local material orientation is created in the full three-dimensional model based on the corresponding orientation definition in the partial three-dimensional model. However, if the material does not contain an orientation definition in the partial three-dimensional model and the partial three-dimensional model is not created by revolving an axisymmetric model, no local orientation definition is created in the full three-dimensional model. The full three-dimensional model uses a global coordinate system.

When results are transferred from a three-dimensional sector to a periodic three-dimensional model by revolving the three-dimensional sector about its symmetry axis, a local cylindrical orientation system is always used for element output of stress, strain, etc. If an orientation is specified in the original three-dimensional sector, the orientation system in the new model is defined by revolving the original orientation system about the symmetry axis. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. If the material in the original three-dimensional sector does not contain an orientation definition, a default local orientation definition is provided. This default orientation is defined by revolving the global coordinate system in the original model about the axis of symmetry in the new model.

Coordinate System at Nodes

The displacement and rotational components obtained from the original model are first transformed into a global, rectangular Cartesian axis system before the results are transferred. If local coordinate directions are required in the new model, a nodal transformation (*Transformed Coordinate Systems*) must be specified in the new model to define this coordinate system.

Limitations

The following limitations exist for result transfer from an axisymmetric model to a 3D model:

- Result transfer is not available from 8-node reduced-integration axisymmetric elements (CAX8R and CAX8RH) to the corresponding 20-node brick elements (C3D20R and C3D20RH) when the elements are underlying the secondary surface in a contact pair.
- SAX2 is a finite-strain shell, while S8R is a small-strain shell. Do not use this combination when deformations are large in the original analysis.

The following limitation exists for result transfer from a symmetric 3D model to a full 3D model:

- Result transfer is not supported for shells with five degrees of freedom per node (STRI65, S8R5, and S9R5).

Initial Conditions

Initial conditions (see *Initial Conditions*) can be specified on all nodes and elements, including the part of the model generated using symmetric model generation (see *Symmetric Model Generation*). However, in most cases the symmetric results transfer capability will overwrite the initial condition values with the solution obtained from the original model. An exception is initial temperatures and field variables. Initial temperatures and field variables are overwritten only when temperatures and field variables are specified in the original model. If only part of the original model contains a specification for temperatures or field variables, the remaining part of the model is assumed to have initial conditions with a magnitude of zero. This distribution of the field will be transferred to the new model. If temperatures and/or field variables are not defined anywhere in the original model, the initial conditions specified in the new model are applied.

Boundary Conditions

All boundary conditions must be redefined; the symmetric result transfer capability ignores the boundary conditions specified in the original model. You must ensure that the model is in equilibrium at the beginning of the analysis; therefore, an initial step definition should be included using boundary conditions and loading that match the state of the model from which the results are transferred.

Loads

All loads must be redefined; the symmetric result transfer capability ignores the loads specified in the original model. You must ensure that the model is in equilibrium at the beginning of the analysis; therefore, an initial step definition should be included using boundary conditions and loading that match the state of the model from which the results are transferred.

Material Options

All of the material definitions defined in the original model will be transferred to the new model.

Elements

Any element or contact pair removal/reactivation definition (see *Element and Contact Pair Removal and Reactivation*) that was active in the original model should be respecified.

Output

All of the standard output variables available for stress/displacement elements can be used with the symmetric results transfer capability.

The solution that is transferred to the new model can be written to the results (`.fil`) file by requesting output at the beginning of a step (the zero increment; see *About Output*). It can also be displayed in Abaqus/CAE.

Analysis of Models that Exhibit Cyclic Symmetry

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

Overview

The cyclic symmetry analysis technique:

- makes it possible to analyze the behavior of a 360° structure with cyclic symmetry based on a model of a repetitive sector;
- can determine the response to cyclic symmetric loading in static, quasi-static, explicit dynamic, and heat transfer analyses;
- in Abaqus/Standard, can calculate all eigenfrequencies and eigenmodes of the 360° structure with the block Lanczos eigenfrequency extraction procedure;
- in Abaqus/Standard, can determine the response to loading corresponding to a given cyclic symmetry mode in modal-based steady-state dynamic analysis; and
- does not require that matched meshes be used on the symmetry surfaces.

References:

- *Natural Frequency Extraction*
- *Mode-Based Steady-State Dynamic Analysis*
- **CYCLIC SYMMETRY MODEL*
- **SELECT CYCLIC SYMMETRY MODES*
- **SURFACE*
- **TIE*
- *Defining cyclic symmetry*

Introduction

Structures that exhibit cyclic symmetry provide the analyst with an opportunity to model an entire 360° structure at considerably reduced computational expense by analyzing only a single repetitive sector of the model. Typically, this is the smallest sector that can be identified, although this is not necessary. For example, if a structure consists of 16 repetitive sectors it is possible to use a 45° model containing two repetitive sectors. The sectors are numbered in the counterclockwise direction to the axis of cyclic symmetry (as described further below). Of course this is less efficient than using a 22.5° model with one sector. There is no restriction that the meshes on the two symmetry surfaces of a repetitive sector match in any way.

There are two basic cases that must be considered in such an analysis: a model that has a cyclic symmetric initial state and a cyclic symmetric response, and a model with a cyclic symmetric initial state but a nonsymmetric response. The cyclic symmetry capability provides for linear and nonlinear analysis of cyclic symmetric structures with cyclic symmetric response. The condition that the structure be cyclic symmetric holds throughout the analysis, so in a loading step it is not possible to have any nonsymmetric deformation in the structure at any time. Therefore, only cyclic symmetric loads can be applied for this situation.

Analysis of cyclic symmetric structures that exhibit nonsymmetric response requires additional consideration. Such an analysis can be performed only in Abaqus/Standard and only in a linear perturbation step, since the nonsymmetric deformation invalidates the assumption of a cyclic symmetric “base state” for any subsequent step in a general nonlinear analysis. The full response of an entire cyclic symmetric structure, such as the structure

illustrated in *Figure 1*, can be represented as a linear combination of several independent basic responses, each of which corresponds to some *k*-fold cyclic symmetry mode.

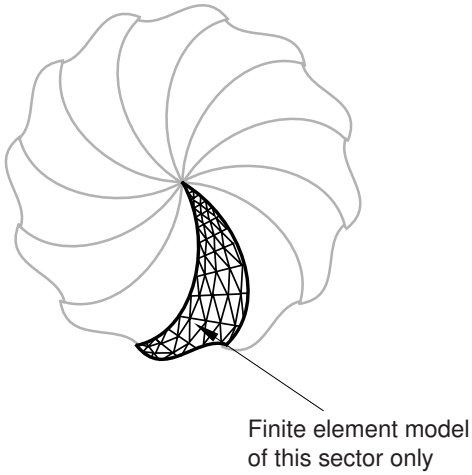


Figure 1: Cyclic symmetric structure.

The cyclic symmetry mode number, which is sometimes also referred to as the "nodal diameter," indicates the number of waves along the circumference in a basic response. *Figure 2*, *Figure 3*, and *Figure 4* illustrate basic responses corresponding to the 0-, 1-, and 2-fold modes (nodal diameters 0, 1, and 2) in a cyclic symmetric structure containing four repetitive sectors. A full linear perturbation analysis can be performed by solving a sequence of corresponding linear analyses for a symmetric single sector. Cyclic symmetric boundary conditions (associated with various cyclic symmetry modes) on the single sector give rise to Hermitian stiffness and mass matrices (complex matrices with symmetric real parts and skew-symmetric imaginary parts). The *k*th linear analysis in the sequence is performed using symmetry conditions that correspond to the *k*-fold cyclic symmetry mode of the structural response. For a structure exhibiting *N*-fold cyclic symmetry, only $\boldsymbol{N/2+1}$ (*N* even) or $\boldsymbol{(N+1)/2}$ (*N* odd) such analyses are required. This results in a solution for the response of the entire structure at a relatively low computational expense.

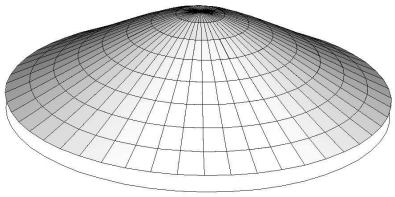


Figure 2: Response corresponding to the 0-fold cyclic symmetry mode.

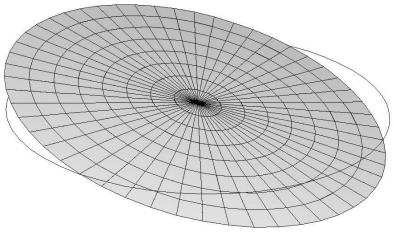


Figure 3: Response corresponding to the 1-fold cyclic symmetry mode.

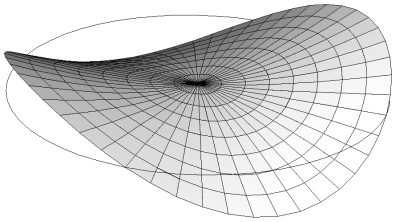


Figure 4: Response corresponding to the 2-fold cyclic symmetry mode.

To perform a general linear analysis of a cyclic symmetric structure, the external forces should be represented as a linear combination of basic loads, each of which corresponds to a symmetry mode and excites a structural response corresponding to the same mode. In static analysis a capability to define loads on any mode other than the 0-fold mode has not yet been implemented. As the response of the 0-fold mode preserves cyclic symmetry, analysis of this type of structure can be done in a general nonlinear step, as well as in a linear perturbation step (as described above). For the same reason, such a step can be used as a preload step for a cyclic symmetric linear perturbation step.

Extraction of a nonsymmetrical response for a cyclic symmetric structure is currently available only for eigenfrequency extraction analysis (*Natural Frequency Extraction*) using the block Lanczos method and for frequency domain, modal-based steady-state dynamic analysis (*Mode-Based Steady-State Dynamic Analysis*). Natural frequencies corresponding to both symmetric and nonsymmetric eigenmodes can be extracted for a specific cyclic symmetry mode, for a group of cyclic symmetry modes, or for all cyclic symmetry modes. They can be used within the subsequent steady-state dynamic analysis. The eigenmodes onto which the solution is projected are chosen as described in *Selecting the Modes and Specifying Damping*.

In a steady-state modal-based dynamic analysis, concentrated, distributed, and surface loads can be defined as projected onto a specific cyclic symmetry mode. Within the same steady-state dynamics step all applied loads have to be given as projected onto the same cyclic symmetry mode. This limitation implies that the specified cyclic symmetry mode must be the same for all loads within the given steady-state dynamics step.

Defining a Cyclic Symmetric Model

Define the mesh for a single sector of the model, the so called “datum sector.” Specify the number of sectors, n, in the 360° model. Define the axis of symmetry by specifying the coordinates (in the global coordinate system) of two points lying on the axis. The axis direction is from the first point to the second point, and the sectors are numbered counterclockwise around the axis, with the datum sector as sector number 1. For a two-dimensional

model only a single point needs to be given on the axis. The axis direction is assumed to be in the positive z-direction; hence, the sectors are numbered counterclockwise in the x–y plane.

Input File Usage: **CYCLIC SYMMETRY MODEL*, N=*n*

In a model defined in terms of an assembly of part instances, the **CYCLIC SYMMETRY MODEL* option must appear within the model definition (see *Assembly Definition*).

Abaqus/CAE Usage: Interaction module: **Interaction->Create**: **Cyclic symmetry**: **Total number of sectors**: *n*

Applying Cyclic Symmetry Constraints

To apply the cyclic symmetry constraints, you must define one or more pairs of corresponding surfaces on each side of the datum sector (see *About Surfaces*). You can then apply the cyclic symmetry constraints between the pairs of corresponding surfaces using a cyclic symmetry surface-based tie constraint (see *Defining Tied Contact in Abaqus/Standard*). The first surface of each pair specified in the tie constraint definition is the secondary surface, and all degrees of freedom of the nodes in the surface will be eliminated by internally generated multi-point constraints. The second surface of each pair is a main surface. If more than one pair of secondary/main surfaces is defined, the rotation direction from the secondary surface to the main surface must be the same for all pairs (i.e., clockwise or counterclockwise).

Input File Usage: Use the following options to apply a cyclic symmetry constraint between two surfaces:

**SURFACE*, NAME=*main*
**SURFACE*, NAME=*secondary*
**TIE*, CYCLIC SYMMETRY, NAME=*cyclic*
secondary, main

Abaqus/CAE Usage: Interaction module: **Interaction->Create**: **Cyclic symmetry**: click **Surface** in the prompt area

Using Mismatched Surface Meshes

In the case of mismatched surface meshes, as shown in *Figure 5*, the finer mesh should typically be the secondary surface. Mismatched meshes may cause some local inaccuracies in the stress field. The magnitude of the inaccuracies depends on the degree of mismatch between the meshes as well as on the element type used: the inaccuracies are typically most pronounced for second-order (modified) tetrahedral elements. Hence, if mismatched surface meshes are used, it is recommended that the sector boundaries be chosen in areas where accuracy of the local stress field is not critical.

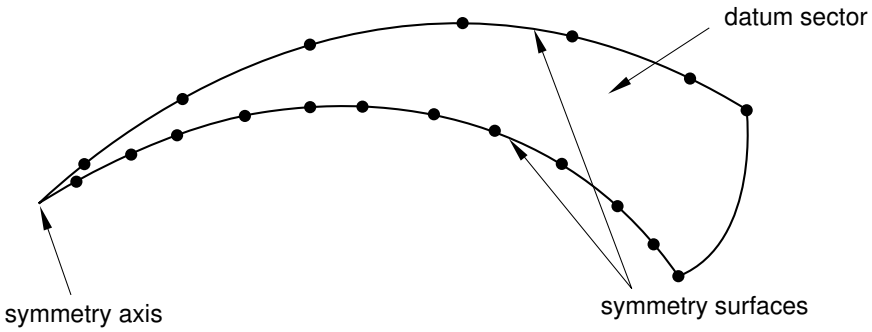


Figure 5: Cyclic symmetry surfaces with mismatched nodes.

For shells the cyclic symmetry condition has to be applied to the nodes on the edges of the shell elements. Currently cyclic symmetry is not supported for element-based surfaces defined on the edges of shells. Therefore, if mismatched meshes are used for shell elements, an element-based surface should be defined on the top or bottom of the shell elements adjacent to the edges that form the main surface. A node-based surface can be defined on the edge that forms the secondary surface.

Applying Node-to-Node Cyclic Symmetry Constraints

In the case of matched meshes, either surface can be chosen as the secondary surface. If the surfaces have matched meshes, as shown in *Figure 6*, it is possible to use a node-based main surface to obtain node-to-node cyclic symmetry constraints. The advantage of this is that Abaqus/Standard will adjust the positions of the nodes on the secondary surface so that they precisely match the positions of the nodes on the main surface. This yields the most accurate results and minimizes the computational cost. In this case the secondary surface will typically be chosen as a node-based surface as well, although computationally it does not matter since in either case a strict node-to-node constraint is applied.

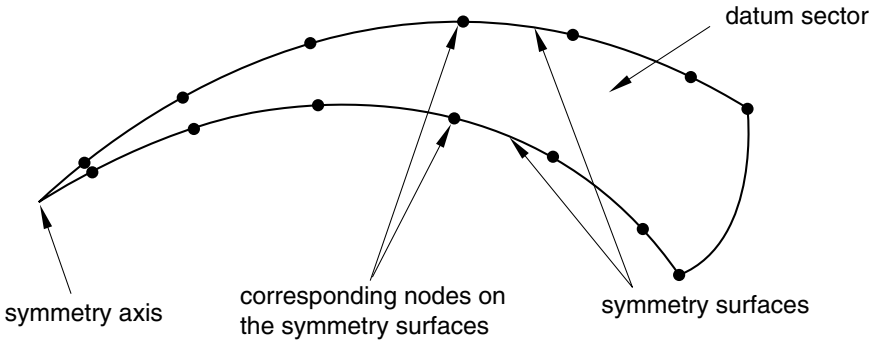


Figure 6: Cyclic symmetry surfaces with node-to-node matching.

For discrete members (such as trusses or beams) the cyclic symmetry condition can be enforced only using node-based surfaces.

Input File Usage: Use the following options to apply a cyclic symmetry constraint between two node-based surfaces:

**SURFACE*, TYPE=NODE, NAME=*main*
**SURFACE*, TYPE=NODE, NAME=*secondary*
**TIE*, CYCLIC SYMMETRY, NAME=*cyclic*
secondary, main

Abaqus/CAE Usage: Interaction module: **Interaction->Create**: **Cyclic symmetry**: click **Node Region** in the prompt area

Applying Cyclic Symmetry Conditions on the Symmetry Axis

If a node is located on the symmetry axis, special cyclic symmetry constraints must be applied for the 0-fold and 1-fold cyclic symmetry modes; whereas all degrees of freedom must be constrained for the other cyclic symmetry modes. For the 0-fold cyclic symmetry mode the degrees of freedom in the plane orthogonal to the symmetry axis are constrained; for the 1-fold cyclic symmetry mode the degrees of freedom along the symmetry axis are constrained. Abaqus/Standard will create these constraints automatically as long as the node is included in the definition of the secondary surface, the main surface, or both the secondary and main surfaces.

Obtaining All Eigenfrequencies of a Cyclic Symmetric Structure

The natural frequencies and corresponding eigenmodes of a cyclic symmetric structure can be extracted using the eigenfrequency extraction procedure with the Lanczos eigensolver (see *Natural Frequency Extraction*). No additional information is required for the eigenfrequency extraction procedure. All the natural frequencies are sorted in the conventional (ascending) order. For each natural frequency the cyclic symmetry mode number is reported.

The eigenmodes are written in the order corresponding to natural frequencies to the data (`.dat`), results (`.fil`), and output database (`.odb`) files for the user-specified datum sector only. These modes can be expanded in Abaqus/CAE to the entire structure depending on the cyclic symmetry mode number.

There are two different types of eigenmodes: single and paired. The eigenmodes for 0-fold cyclic symmetry are always single. For even N the eigenmodes for the $N/2$-fold cyclic symmetry are also single. The eigenmodes for the remaining $N/2-1$ (even N) or $(N-1)/2$ (odd N) cyclic symmetry modes are paired. The natural frequencies corresponding to the paired eigenmodes are equal and always appear together in the table of the natural frequencies in the data file. The expansion of the eigenmodes with k-fold cyclic symmetry ($k < N/2$) to the sector $i = 2, \ldots, N$ can be done in the following manner:

$$\begin{bmatrix} u_k^i \\ v_k^i \end{bmatrix} = [T_k^i] \begin{bmatrix} u_k \\ v_k \end{bmatrix},$$

where

$$[T_k^i] = \begin{bmatrix} cos\Psi_k (i-1) & sin\Psi_k (i-1) \\ -sin\Psi_k (i-1) & cos\Psi_k (i-1) \end{bmatrix}.$$

Here (u, v) and (u^i, v^i) are paired eigenmodes corresponding to double natural frequencies on the first (datum) sector and on the ith sectors, respectively; and $\Psi_k = 2\pi k/N$.

From the expressions above it is clear that eigenmodes with 0-fold cyclic symmetry are always symmetric; i.e., $u_0^i = u_0$. Similarly, for even N the eigenmodes with $(N/2)$-fold cyclic symmetry are single, since $u_{(N/2)}^i = (-1)^{(i-1)} u_{(N/2)}$.

Selecting the Cyclic Symmetry Modes

You can select the cyclic symmetry modes for which the eigenfrequency analysis will be performed by specifying the lowest cyclic symmetry mode to be used in the analysis, *nmin*, and the highest cyclic symmetry mode to be used in the analysis, *nmax*. By default, *nmin* is 0. By default, *nmax* is $N/2$ (even N) or $(N-1)/2$ (odd N). The

value of *nmin* cannot be greater than the value of *nmax*, and the value of *nmax* cannot be greater than the default value. If you do not select the cyclic symmetry modes, all possible cyclic symmetry modes are considered in the analysis. You can choose to use only the even cyclic symmetry modes.

Input File Usage: Use the following option to specify the cyclic symmetry modes:

**SELECT CYCLIC SYMMETRY MODES*, NMIN=*nmin*, NMAX=*nmax*

Use the following option to request only the even cyclic symmetry modes:

**SELECT CYCLIC SYMMETRY MODES*, EVEN

Abaqus/CAE Usage: Use the following option to specify the cyclic symmetry modes:

Interaction module: **Interaction->Create**: **Cyclic symmetry**: toggle on **Specified range** and specify the **Lowest nodal diameter** and **Highest nodal diameter**

You cannot request only the even cyclic symmetry modes in Abaqus/CAE.

Selecting the Cyclic Symmetry Mode for a Steady-State Dynamic Step

Only a single cyclic mode can be excited in a steady-state dynamic step. You specify the cyclic symmetry mode associated with the loading in the load definition.

Input File Usage: Use one of the following options:

**CLOAD*, CYCLIC MODE=*k*, REAL or IMAGINARY
**DLOAD*, CYCLIC MODE=*k*, REAL or IMAGINARY
**DSLOAD*, CYCLIC MODE=*k*, REAL or IMAGINARY

Abaqus/CAE Usage: Interaction module: **Interaction->Create**: **Cyclic symmetry**: **Excited nodal diameter**

Comparison of the Cyclic Symmetry Analysis Technique and MPC Type CYCLSYM

MPC type CYCLSYM (*General Multi-Point Constraints*) provides a subset of the functionality provided by the cyclic symmetry analysis capability. For an eigenvalue analysis MPC type CYCLSYM will allow extraction of the symmetric (0-fold) modes only. The cyclic symmetry analysis capability allows the use of surfaces (*About Surfaces*) to define the symmetry surfaces for the model, which enables the use of mismatched meshes on the symmetry surfaces, whereas MPC type CYCLSYM can be applied only on a node-to-node basis.

Limitations

The following limitations exist:

- A continuation capability is not available for the cyclic symmetry eigenvalue extraction procedure. Each eigenvalue extraction step will not reuse any eigenmodes obtained in the previous eigenvalue extraction steps.
- The specified cyclic symmetry mode must be the same for all loads defined within a given steady-state dynamic step.
- Base motion is not implemented for cyclic symmetry models.

- Cyclic symmetry conditions are applied to the mechanical degrees of freedom in stress/displacement analysis and temperature degrees of freedom in heat transfer analysis. Cyclic symmetry conditions are not applied to acoustic pressure, pore pressure, and electrical degrees of freedom.
- You cannot use cavity radiation in cyclic symmetric models.
- You cannot use a dynamic temperature-displacement analysis in cyclic symmetric models.

Initial Conditions

All applied initial conditions must be cyclic symmetric.

Boundary Conditions

Only cyclic symmetric boundary conditions can be applied. Boundary conditions cannot be applied to the nodes on the secondary cyclic symmetry surface.

Loads

In static analysis only cyclic symmetric loads can be applied. Coriolis loads cannot be applied, and the effect of the Coriolis load stiffness is not considered in the frequency analysis.

In modal-based steady-state dynamic analysis the loads are defined on the datum sector for a specific cyclic symmetry mode, which is indicated in the loading definition. For the *k*-fold cyclic symmetry mode $(0 < k < N/2)$ the complex loads f^i and g^i (corresponding to real and imaginary components, respectively) on the sector $i = 1, 2, \ldots, N$ are obtained in the following manner:

$$f^i = Fcos\Psi_k\,(i-1) + Gsin\Psi_k\,(i-1)\,,$$
$$g^i = Gcos\Psi_k\,(i-1) - Fsin\Psi_k\,(i-1)\,,$$

where $\Psi_k = 2\pi k/N$ and *F* and *G* are real and imaginary components of loads specified for the datum sector, respectively. For the 0-fold cyclic symmetry mode ($k = 0$) this type of loading corresponds to a cyclic symmetric load pattern with $f^i = F$ and $g^i = G$. For $k > 0$ this type of loading is generated when a spatially constant load pattern is applied to a rotating structure (or when a constant load pattern rotates around the structure). For the $N/2$-fold mode the complex loads on the sector *i* are: $f^i = (-1)^{(i-1)}F$ and $g^i = (-1)^{(i-1)}G$.

Predefined Fields

Only cyclic symmetric predefined fields can be applied. Hence, the predefined fields should have the same values at each side of the datum sector.

Material Options

No specific restrictions apply to material models for cyclic symmetry models of general procedures. For the frequency analysis procedure, see the remarks in *Natural Frequency Extraction*.

Elements

Axisymmetric elements should not be used in cyclic symmetry models.

Output

Nodal displacements and element output variables such as stress, strain, and section force are available only for the datum sector. The mass listed in the data file is computed for the whole model.

The following conditions apply in the eigenvalue extraction procedure:

- If you select the displacement eigenvector normalization (default), the largest displacement entry in each eigenvector on the datum sector is unity. If you select the mass eigenvector normalization, the eigenvectors are normalized so that the generalized mass computed for the entire model is unity. For more information, see *Natural Frequency Extraction*.
- The eigenvalue numbers, cyclic symmetry mode numbers, and corresponding frequencies (in both radians/time and cycles/time) are listed in the data file, along with the generalized masses, composite modal damping factors, participation factors, and modal effective masses. The generalized masses, composite modal damping factors, participation factors, and modal effective masses are calculated for the entire model.
- You can restrict output to the results and data files by selecting the modes for which output is desired (see *Output to the Data and Results Files*).
- With Abaqus/CAE static displacements and eigenmodes can be displayed for any sector. The results of steady-state, modal-based dynamic analysis can also be animated for any number of sectors, including the entire model.

Input File Template

Input file templates for creating a cyclic symmetry model for an Abaqus/Standard analysis and for an explicit dynamic analysis are available.

The following template can be used to create a cyclic symmetry model in an Abaqus/Standard analysis:

```
*HEADING
…
**
*CYCLIC SYMMETRY MODEL, N=integer
N denotes the number of sectors in the entire 360° model.
…
**
*SURFACE, NAME=name, TYPE=ELEMENT
*SURFACE, NAME=name, TYPE=NODE
Surface description for the secondary and main nodes that will be referenced in the *TIE option.
…
**
*TIE, CYCLIC SYMMETRY
Indicates the internal MPCs that tie the main and secondary surfaces
using the cyclic symmetry condition in the cyclic symmetry models only.
Data lines to specify surface names that will be tied with this option.
…
**
*STEP (,NLGEOM)
If NLGEOM is used, initial stress and preload stiffness effects
will be included in subsequent linear perturbation steps, including the
frequency extraction step
*STATIC
...
*DLOAD
Data lines to specify element or element set, load type, value, (direction).
...
**
*END STEP
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS
…
*SELECT CYCLIC SYMMETRY MODES, NMAX=integer, NMIN=integer, EVEN
```

```
…
**
*END STEP
*STEP
*STEADY STATE DYNAMICS
…
*SELECT EIGENMODES
Use this option to specify the list of eigenmodes used in the response.
*MODAL DAMPING
Data lines to specify damping coefficients associated with eigenmodes.
…
*CLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
Data lines to specify node or node set, degree of freedom, value
*DLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
Data lines to specify element or element set, load type, value, (direction)
…
*DSLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
Data lines to specify element or element set, load type, value, (direction)
…
**
*END STEP
```

The following template can be used to create a cyclic symmetry model for an explicit dynamic analysis:

```
*HEADING
…
**
*CYCLIC SYMMETRY MODEL, N=integer
…
*TIE, CYCLIC SYMMETRY
…
*STEP
*DYNAMIC, EXPLICIT
*END STEP
```

Periodic Media Analysis

Products: Abaqus/Explicit

References:

- **PERIODIC MEDIA*
- **MEDIA TRANSPORT*

Overview

The periodic media analysis technique in Abaqus/Explicit:

- is a Lagrangian technique that offers an Eulerian-like view into a moving structure;
- can be used to effectively model systems that are repetitive in nature, such as manufacturing processes involving conveyor belts or continuous forming operations;
- leads to significant analysis time speedup when compared to traditional modeling techniques that may require excessively large meshes; and
- requires topologically identical meshed parts to create the model, which can be accomplished via the parts and instances modeling paradigm.

Introduction

Quite often industrial processes that need to be analyzed involve sections that repeat in a simple pattern and move through a process zone. A prominent example is a conveyor belt with regularly spaced packages, as illustrated schematically in *Figure 1* and exemplified with a finite element mesh in *Figure 2*. Continuous forming operations such as metal rolling are also good examples because the deforming material can be broken up into an arbitrary number of identical sections.

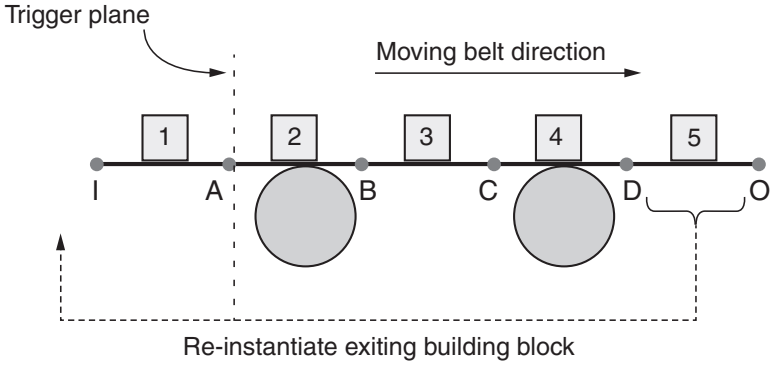


Figure 1: Schematic representation of periodic media.

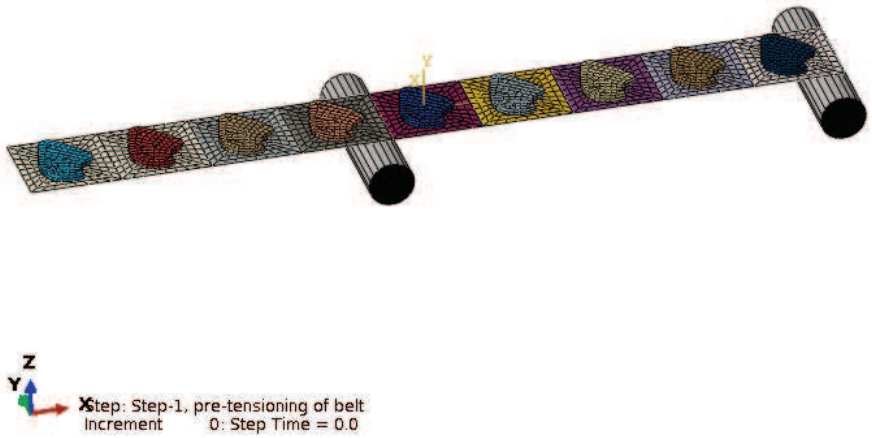


Figure 2: Conveyor belt with packages on top.

For the sake of clarity we will use the conveyor belt example throughout this discussion to illustrate many of the concepts associated with the periodic media analysis technique. *Figure 1* shows a conceptual decomposition of the conveyor belt; in reality, the belt is a continuous entity.

Conceptually, the overall model can be decomposed into blocks (topologically identical meshed structures) that are connected together and span the process zone. You create a part that defines a "building block" (the meshed structure that is repeated to model the entire periodic media) and then construct the whole model via a chain of appropriately positioned instances. The periodic media analysis technique provides a simple way to automatically connect these instances together at the front and back ends of adjacent blocks. This technique also provides a convenient way to define loads and boundary conditions that represent the physical system at the unconnected ends of the first and last blocks in the chain. The first block of the chain is referred to as the inlet, and the last block is referred to as the outlet. Finally, when the periodic media moves through the process zone, blocks from the outlet are automatically shuffled to the inlet. The blocks (meshed structures) defined with this technique can interact via contact with other modeling features that are not periodic in nature, such as the rollers depicted in *Figure 1*.

At the core of the periodic media analysis technique lies the concept of shuffling blocks from the outlet back to the inlet. A dedicated algorithm is used to detect when the inlet has moved too far into the process zone and to shuffle a block from the outlet directly to the inlet. The dashed arrow in *Figure 1* illustrates the shuffling process. To ensure a smooth transition, the necessary nodal and element state data from the inlet block are stored at the beginning of the current step. When shuffling occurs, the stored nodal and element state data are mapped to the new inlet block and any inlet/outlet loads or boundary conditions are transferred to the newly exposed block ends.

Thus, the periodic media analysis technique offers a convenient way for an Eulerian-like view into the moving repetitive structure. For example, you may be interested in assessing the package dynamics on the belt at a location somewhere between the rollers in both transient and steady-state conditions. You define several blocks around that location, you define contact with the rollers as necessary, and you provide appropriate inlet and outlet loading conditions. The periodic media analysis technique provides a convenient and economical way to create and analyze this system. By re-using elements that have left the process zone via this shuffling process, you can avoid the large meshes at the inlet end required for purely Lagrangian simulations.

Constructing a Periodic Media Model

The first step in constructing a periodic media model is to identify the portion of the model that constitutes the building block of the repetitive structure. In *Figure 2* one square belt patch together with one asymmetrically shaped package on top constitute such a building block. If you string together several blocks, the entire belt with packages can be modeled as shown.

Defining a Building Block

The following requirements must be observed when defining each building block:

- an unsorted element set must be defined to include all elements in the building block, and
- an unsorted node set must be defined to include all nodes in the building block.

To ensure the proper transfer of information as the periodic media advances, these unsorted sets must be topologically identical between all blocks. The easiest way to achieve this requirement is to use the parts and instances modeling paradigm. You define one part corresponding to the building block and define unsorted element and node sets as discussed above. You then instantiate the part as many times as needed with the appropriate translations and rotations to generate the periodic media mesh. Constraints such as ties, couplings, and rigid bodies are allowed within a building block. You must ensure that these constraints are defined in a topologically identical fashion in all blocks.

The periodic media analysis technique connects together these otherwise unconnected blocks to create a continuous model. If structural elements (e.g., shells) are used in the connecting regions of the blocks, the nodes on the edges of these regions are connected to the adjacent regions. If continuum elements are used, the nodes on the faces of these regions are connected. For these constraints to be constructed reliably, the following additional requirements must be observed:

- the nodal arrangements at the front and back connecting ends of blocks must be topologically identical,
- the front and back end nodes of adjacent blocks must be coincident,
- the nodal arrangements at the front and back end of the initial inlet block must have coordinates that differ only by a rigid body translation, and
- two node-based surfaces created using unsorted node sets at the front and back end of each block must be defined.

The node-based surfaces are used to automatically generate node-to-node tie constraints between adjacent blocks such that the whole assembly behaves as a continuous entity.

Input File Usage: Use the following option to define the sequence of blocks using unsorted sets and surfaces as described above:

**PERIODIC MEDIA*, NAME=*name*
elset1, *nodeset1*, *frontsurf1*, *backsurf1*
elset2, *nodeset2*, *frontsurf2*, *backsurf2*
...
elsetn, *nodesetn*, *frontsurfn*, *backsurfn*

Each data line provides the set and surface names associated with a given block.

Applying Loads and Boundary Conditions at Media Ends

In the schematic belt shown in *Figure 1* you usually need to apply loads or boundary conditions at both ends of the assembly. At the inlet point I it is often useful to apply a pre-tension load that keeps the belt taut, while at the outlet point O the belt velocity is usually prescribed. As the belt advances and exiting blocks are being shuffled from the outlet to the inlet, the nodes requiring the boundary conditions will change. Therefore, these boundary conditions and loads cannot be prescribed directly at nodes belonging to the block.

The periodic media analysis technique allows for the application of such loading features via two control nodes that are associated with the current inlet and outlet node-based surfaces. The control nodes are similar to reference nodes used in other features (such as kinematic couplings) and impose automatically defined rigid body–like constraints on the nodes at the extreme ends of the assembly. You apply loads and boundary conditions at these

control nodes. A rigid body–like constraint is also imposed on the front end nodes of the inlet block, but no loads or boundary conditions can be applied during shuffling. When exiting blocks are being shuffled back to the inlet, the control points will enforce these rigid body–like constraints on the new extreme end surfaces and remove the rigid body–like constraints from the previous locations. The process is automatic and fully managed by the periodic media analysis technique.

Since the initial configuration of the very first inlet block at the beginning of the shuffling step will be used as a template to initialize the blocks when they are shuffled back to the inlet end, the block should be kept inline with the moving direction to prevent drifting of the inlet edge during shuffling. We recommend that you apply boundary conditions at the inlet control node and front end nodes in the pre-stretching (no block shuffling) step to prevent movement in the direction perpendicular to the moving direction in the shuffling step.

Input File Usage: Use the following option to define control nodes for the inlet and outlet conditions:

**PERIODIC MEDIA*, INLET CONTROL NODE=*node*,
OUTLET CONTROL NODE=*node*

Defining the Process Zone

When the inlet block moves completely into the process zone, the outlet block is shuffled back to the inlet, as the dashed arrow indicates in *Figure 1*. A trigger plane controls the precise timing for when the shuffling occurs. When the nodes located at the current inlet point I cross the trigger plane, the shuffling process is launched. The trigger plane is defined using the coordinates of a (usually) stationary node and the z-axis of a user-defined orientation. The local z-axis direction points from the inlet toward the process zone.

Input File Usage: Use the following option to define the trigger plane via a trigger node and orientation:

**PERIODIC MEDIA*, TRIGGER NODE=*node*,
ORIENTATION=*orientation*

Activating a Periodic Media

The shuffling process can be activated on a step-by-step basis. By default, the shuffling process is inactive. In many cases the configuration of the periodic media in the operating condition can be determined only via simulation. This allows any number of analysis steps to be carried out prior to activating the shuffling process.

The example illustrated in *Figure 2* and in *Media transport* shows a conveyor belt transporting asymmetrical packages placed initially at regular intervals. In its operating condition the belt will be tensioned. You can pre-stretch the belt assembly in either Abaqus/Standard or Abaqus/Explicit. If the pre-stretch analysis is conducted in Abaqus/Standard, all ties between adjacent blocks as well as boundary conditions at the inlet and outlet ends nodes need to be defined explicitly as the periodic media analysis technique is available only in Abaqus/Explicit. If the pre-stretching step is conducted in Abaqus/Explicit, the shuffling process should remain inactive during the pre-stretching step. A boundary condition should be applied at the front end of the inlet block to keep the inlet block aligned with the direction of the movement in the pre-stretching step, and this boundary condition should be removed after the shuffling process is activated in the following step.

Input File Usage: Use the following option to activate or deactivate the periodic media shuffling process:

**MEDIA TRANSPORT*
periodic_media_name1, ACTIVE
periodic_media_name2, INACTIVE
...

Modeling Tips

The periodic media analysis technique is a powerful feature; however, you must exercise good engineering judgement when using it. The following comments and recommendations will help you avoid common pitfalls when using this technique:

- The block shuffling process is inherently noisy as chunks of elements are detached at one end and reattached at the other. Although the process uses appropriate material and kinematic states, small shocks are inherent to the process. A small amount of mass proportional damping is recommended to dampen out this excitation.
- The combination of boundary conditions at the inlet control node and any loads applied in the process zone should ensure that the inlet block moves across the trigger plane without a change in direction. In the conveyor belt example, a good modeling practice would be to place a fixed guide roller at least two blocks away from the trigger plane.
- For more complex geometries (such as belts that change direction between rollers or package wrapping analyses when the belt is the wrapping material itself), it may be necessary to start with a straight sequence of blocks and move the belt rollers (which are not part of the periodic media definition) into the desired locations. Contact interaction between the belt and the rollers would deform the belt in the desired configuration. This additional analysis step can greatly simplify the definition of the initial mesh.
- Sometimes it may be necessary to model the process of threading a belt wrapping through rollers, just as in physical reality at the start of a manufacturing process. If this leading segment is followed by periodic blocks that include actual packages, you can attach the periodic media mesh to a regular mesh to execute the threading. The periodic media part of the mesh can then be imported into a separate model without the leading mesh, and the analysis of the periodic media consisting only of the wrapper and packages can be executed.

Initial Conditions

Initial conditions can be specified at all nodes in the periodic media mesh. Velocity initial boundary conditions can be used to minimize the solution time needed to reach a steady-state operating condition. In cases where pre-stretching is required, importing from the prior analysis rather than performing a multistep analysis allows for initial conditions to be applied to the stretched configuration. Since periodic media definitions are not imported, they must be respecified in every analysis in which they are required.

Boundary Conditions

The inlet and outlet control nodes are the only two nodes associated with a periodic media definition at which boundary conditions can be specified when it is active. Furthermore, only velocity boundary conditions are permitted. You must not specify boundary conditions at any other node associated with the periodic media mesh when it is active. While the periodic media is active and if a steady-state solution is sought, these boundary conditions should be kept constant in both direction and magnitude to mitigate solution noise.

Loads

Only concentrated loads can be applied to the inlet and outlet control nodes to either drive or stretch the periodic media. While the periodic media is active, these loads should be kept constant in both direction and magnitude. Gravity loads can be applied as desired. Other distributed loads can also be specified; however, you must keep in mind that the loads will travel with the blocks as they are shuffled.

Material Options

All available material models are supported.

Limitations

Periodic media analyses are subject to the following limitations:

- Only membranes, shells, trusses, continuum elements, and rigid elements are allowed within blocks. Rebar layers can also be used, if applicable.
- No explicitly defined constraints are allowed between nodes belonging to different blocks.
- Mass scaling must be defined in the same fashion for all blocks.

The periodic media should not be involved in

- general contact that defines thermal contact properties or coupled Eulerian-Lagrangian contact or
- contact defined via the contact pair algorithm.

Input File Template

The following example illustrates a model with two periodic media defined:

```
*HEADING
…
*PERIODIC MEDIA, NAME=belt1, INLET CONTROL NODE=10,
OUTLET CONTROL NODE=110, ORIENTATION=ori1, TRIGGER NODE=210
elset1, nodeset1, frontedgesurf1, backedgesurf1
elset2, nodeset1, frontedgesurf2, backedgesurf2
elset3, nodeset1, frontedgesurf3, backedgesurf3
*PERIODIC MEDIA, NAME=belt2, INLET CONTROL NODE=11,
OUTLET CONTROL NODE=111, ORIENTATION=ori2, TRIGGER NODE=211
elset1, nodeset1, frontedgesurf1, backedgesurf1
elset2, nodeset1, frontedgesurf2, backedgesurf2
elset3, nodeset1, frontedgesurf3, backedgesurf3
*STEP
*DYNAMIC, EXPLICIT
*MEDIA TRANSPORT
belt1, ACTIVE
belt2, INACTIVE
*END STEP
```

Meshed Beam Cross-Sections

Products: Abaqus/Standard Abaqus/Explicit

Overview

Meshed cross-sections:

- allow for the description of a beam cross-section including multiple materials and complex geometry;
- are meshed in Abaqus/Standard with two-dimensional warping elements, which have an out-of-plane warping displacement as the only degree of freedom for 1-DOF warping elements and two in-plane and one out-of-plane warping displacements as degrees of freedom for 3-DOF warping elements;
- generate beam cross-section properties that can be used in a subsequent beam element analysis in either Abaqus/Standard or Abaqus/Explicit;
- allow only isotropic linear elastic material behavior or orthotropic linear elastic material behavior for 1-DOF warping elements;
- allow isotropic linear elastic material behavior, fully three-dimensional orthotropic linear elastic material behavior, and fully anisotropic linear elastic material behavior for modeling composite beam cross-sections with 3-DOF warping elements; and
- allow stress and strain postprocessing on the beam element model or the two-dimensional warping element model.

References:

- **BEAM GENERAL SECTION*
- **BEAM SECTION GENERATE*
- **SECTION ORIGIN*
- **SECTION POINTS*

Introduction

The response of some structures is beam-like, yet the beam cross-section geometry or multimaterial makeup of the cross-section do not permit the use of a predefined library beam cross-section. In these cases a meshed cross-section can be used to model the beam cross-section and to generate beam cross-section properties appropriate for subsequent use in a Timoshenko beam analysis. The beam properties are generated assuming a thick-walled (solid) cross-section with unconstrained out-of-plane and, optionally, in-plane warping. Therefore, open-section beam elements cannot use the beam cross-section properties generated from the meshed section (see *About Beam Modeling*). The generated beam cross-section properties include axial, bending, torsional, and transverse shear stiffnesses; mass, rotary inertia, and damping properties; and the centroid and shear center of the cross-section. In addition, the equivalent beam cross-section properties include information on stress recovery, such as the warping functions and their derivatives.

A typical example of a structure that requires a meshed cross-section is the hull of a ship for whipping analysis, where the ship's hull has a multi-cell and multimaterial construction. Other examples include an airfoil-shaped rotor blade or wing and a layered composite I-beam (with fibers running along the length of the beam axis or perpendicular to it).

Modeling Approach

As shown in *Figure 1*, a meshed cross-section allows for a complex description of a beam cross-section: one that can include an arbitrary shape, multiple materials, multiple cells, and nonstructural mass. The basic idea is to create a two-dimensional finite element model of the beam cross-section. The meshed cross-section is used in Abaqus/Standard to numerically calculate the properties required to characterize the structural response of the cross-section in a subsequent beam element analysis. The two-dimensional Abaqus/Standard analysis writes the cross-sectional properties to an input-file-ready text file (`jobname.bsp`). In the subsequent Abaqus/Standard or Abaqus/Explicit beam element analysis the beam elements requiring the meshed cross-section properties include the text file `jobname.bsp` as the general beam section data. Once the beam element analysis is complete, the Visualization module of Abaqus/CAE is used to visualize results at preselected points along the beam length or to examine detailed stress and strain results displayed directly on the two-dimensional meshed cross-section.

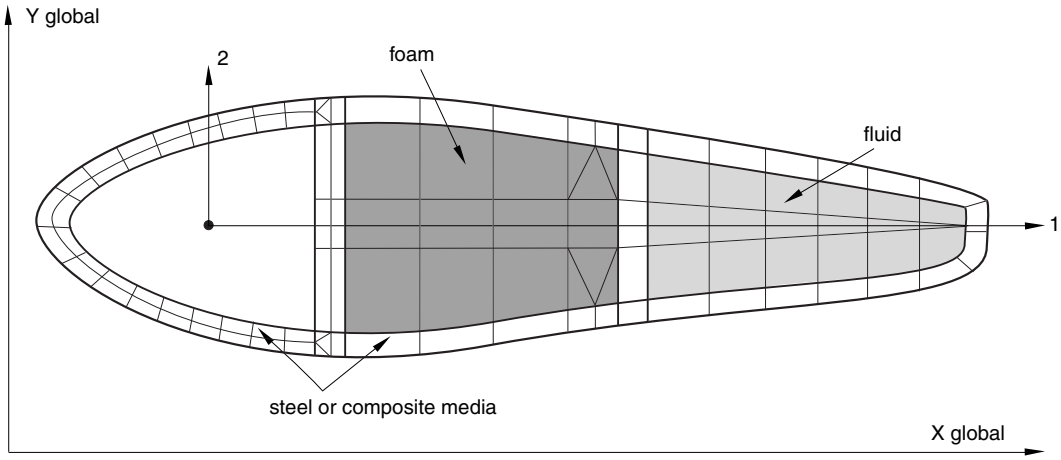


Figure 1: Example of a meshed section profile.

In summary, the procedure for analyzing and postprocessing a beam analysis using a meshed cross-section is as follows:

1. Mesh and analyze a two-dimensional Abaqus/Standard model of the beam cross-section.
2. Use the generated cross-sectional properties in an Abaqus/Standard or Abaqus/Explicit beam analysis.
3. Using the beam analysis results, postprocess from the beam model or the two-dimensional cross-section model.

Meshing and Analyzing a Two-Dimensional Model of the Beam Cross-Section

The cross-section is meshed using special-purpose two-dimensional elements. There are two classes of these warping elements (see *Warping Element Library*):

- WARP2D3 (3-node triangular) and WARP2D4 (4-node quadrilateral) elements have one degree of freedom per node representing the out-of-plane warping displacement.
- WARPF2D3 (3-node triangular), WARPF2D4 (4-node quadrilateral), WARPF2D6 (6-node triangular), and WARPF2D8 (8-node quadrilateral) elements have two additional degrees of freedom per node representing the in-plane warping displacements.

The warping elements use a solid section definition; no section data are required. Adjacent elements in the cross-sectional mesh must share common nodes; mesh refinement using multi-point constraints is not allowed. The mesh can contain either 1-DOF warping elements or 3-DOF warping elements; you cannot mix the two element classes in the same analysis.

Each element in the cross-sectional mesh can refer to a different elastic material. For 1-DOF warping elements, you can use either isotropic (see *Defining Isotropic Elasticity*) or orthotropic linear elastic material behavior (see *Defining Orthotropic Elasticity for 1-DOF Warping Elements*). Alternatively, you can specify density (*Density*) as the only material property, which is useful for modeling nonstructural masses (such as fuel in a tank). For 3-DOF warping elements, the following material behaviors are supported:

- Isotropic linear elastic (see *Defining Isotropic Elasticity*).
- Fully three-dimensional orthotropic linear elastic (see *Defining Orthotropic Elasticity by Specifying the Engineering Constants*, *Defining Transversely Isotropic Elasticity*, and *Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*).
- Fully anisotropic linear elastic (see *Defining Fully Anisotropic Elasticity*).

3-DOF warping elements are particularly useful if you want to model composite beam cross-sections (such as wind turbine rotor blades).

The model is then analyzed by using the beam section property generation procedure within the step definition. This cross-section analysis numerically calculates the geometric, stiffness, and inertial properties of the section, including the warping function and shear center (see *Meshed beam cross-sections*) and writes the calculated properties to the `jobname.bsp` text file. The contents of this text file, which can be used in a subsequent Abaqus/Standard or Abaqus/Explicit beam analysis, are described in detail below.

Input File Usage: Use the following option to generate beam section properties for a meshed cross-section:

**BEAM SECTION GENERATE*

Defining the Origin of the Cross-Section

By default, the origin of the cross-section is the origin of the coordinate system used to define the mesh. You can override this default and input the coordinates of the origin directly or specify that the origin coincides with the shear center or centroid of the cross-section. A nondefault origin is particularly useful when the beam node to be used in the actual analysis does not coincide with the origin of the two-dimensional coordinate system.

Input File Usage: Use both of the following options to input the coordinates of the origin directly:

**BEAM SECTION GENERATE*
**SECTION ORIGIN*

Use both of the following options to locate the origin at either the centroid or shear center:

**BEAM SECTION GENERATE*
**SECTION ORIGIN*, ORIGIN=CENTROID or SHEAR CENTER

Defining the Tangent Direction of the Beam Element Centerline for 3-DOF Warping Elements

The beam centerline is the curve that connects the corner nodes of the beam elements to which you assign the properties that you compute with the beam section property generation procedure. By default, Abaqus assumes that the beam centerline tangent direction is parallel to the cross-section normal direction. For 3-DOF warping elements, you can override this default and specify the direction cosines of the beam centerline tangent. Thereby, if the beam centerline tangent is inclined with respect to the cross-section normal direction, you can take into account the effect of this inclination for the beam element cross-section properties.

Input File Usage: Use both of the following options to specify the direction cosines of the of the beam centerline tangent:

*BEAM SECTION GENERATE
*SECTION TANGENT

Defining the Curvatures and the Torsion of the Beam Element Centerline for 3-DOF Warping Elements

By default, Abaqus assumes that the beam centerline is straight and that beam element cross-sections have the same orientation in space. If this is not the case, 3-DOF warping elements can take into account the influence of the curvatures and torsion of the beam centerline on the stiffness properties of the associated beam element cross-sections. For example, you can model a rotor blade that is prebent and pretwisted.

The beam centerline is the curve that connects the beam element nodes. When you consider the projection of the beam centerline onto the plane that is normal to the cross-section 1-direction (or 2-direction), the associated first (or second) precurvature value is the reciprocal radius of the osculating circles. For example, *Figure 2* shows the osculating circle of the centerline projection onto the plane that is normal to the 1-direction; the associated first precurvature is $\mathbf{1}/\boldsymbol{R}$.

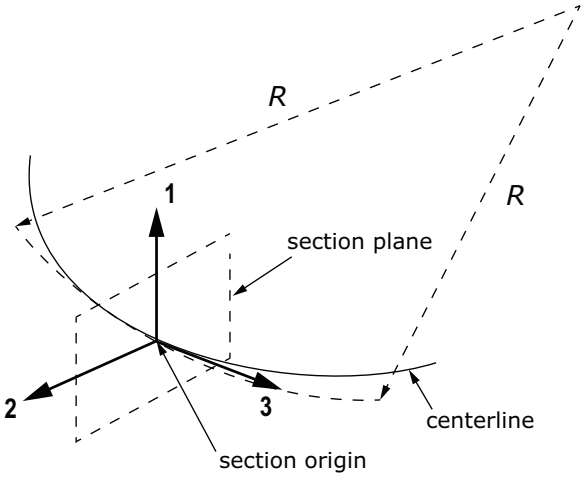


Figure 2: Precurvature.

The pretorsion is defined as the change in rotation of the beam elements' cross-section orientations about the beam centerline tangent with respect to the rotation-minimizing frame along the centerline. For example, in *Figure 3* the centerline is straight and the rotation-minimizing frames in the cross-section planes are all parallel; the associated pretorsion for the depicted cross-section orientations is $\boldsymbol{\gamma}/\boldsymbol{\Delta}$.

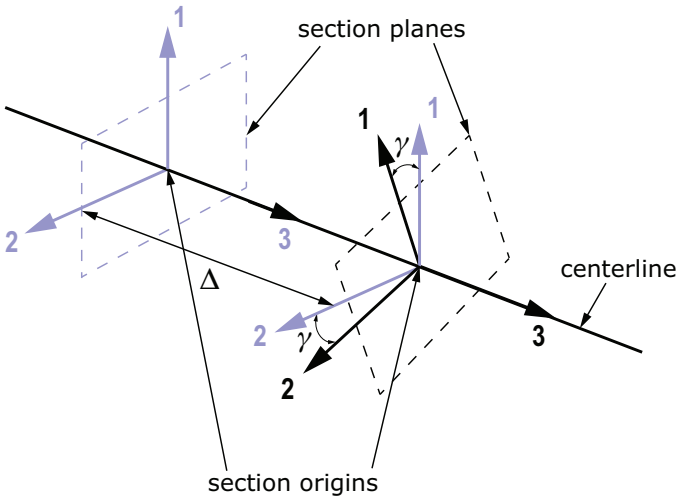


Figure 3: Pretorsion.

Input File Usage: Use both of the following options to specify the beam centerline curvatures and torsion:

**BEAM SECTION GENERATE*
**SECTION CURVATURE*

Requesting Output at Particular Integration Points for 1-DOF Warping Elements

Output to the output database can be recovered during the actual analysis at particular integration points on the cross-section. Requesting output at a large number of cross-sectional points may degrade performance.

Input File Usage: Use both of the following options to request output at particular integration points:

**BEAM SECTION GENERATE*
**SECTION POINTS*

Contents of the Output Database

By default, the output database contains field output with warping displacements for one frame for 1-DOF warping elements or for six frames for 3-DOF warping elements. For 1-DOF warping elements, the field output also contains section point strain and stress recovery data for all element integration points. For 3-DOF warping elements, the strain and stress recovery data is contained as strain and stress element output for the six frames. The data in the first, second, and third frames represent the cross-section warping fields for unit beam cross-section forces due to extension, shearing in the 1-axes, and shearing in the 2-axis of the cross-section, respectively. The data in the fourth, fifth, and sixth frames represent the cross-section warping fields for unit beam cross-section torques due to twisting, bending about the 1-axis, and bending about the 2-axis of the cross-section, respectively.

Contents of the *jobname*`.bsp` Text File

After the analysis to generate the cross-sectional properties completes, the *jobname*`.bsp` text file contains the following lines of data when you use 1-DOF warping elements:

```
*SECTION STIFFNESS
(EA), (EI)11, (EI)12, (EI)22, (GJ)
*SECTION INERTIA
```

(ρA), $(\rho I)_{11}$, $(\rho I)_{12}$, $(\rho I)_{22}$, x_{1cm}, x_{2cm}
*TRANSVERSE SHEAR STIFFNESS
$(GA)_{22}$, $(GA)_{11}$, $(GA)_{12}$
*CENTROID
x_{1c}, x_{2c}
*SHEAR CENTER
x_{1s}, x_{2s}
*DAMPING, ALPHA=(C_α), BETA=(C_β), COMPOSITE=(C_c)

The first two lines of data in the *jobname*`.bsp` text file correspond to the section property data for an arbitrarily shaped solid general beam cross-section meshed with warping elements (see *Defining Linear Section Behavior for Meshed Cross-Sections*).

If you requested output at particular integration points in the two-dimensional cross-section model generation, the *jobname*`.bsp` text file contains the following additional lines:

*SECTION POINTS
section point label, *2D element number*, *integration point number*
E, G_1, G_2, α, $-(x_1 - x_{1c})$, $(x_2 - x_{2c})$, $\left(\frac{\partial \Psi}{\partial x_1} - (x_2 - x_{2s})\right)$, $\left(\frac{\partial \Psi}{\partial x_2} + (x_1 - x_{1s})\right)$
...

where the set of two data lines is repeated for as many section points as requested.

When you use 3-DOF warping elements, the *jobname*`.bsp` text file contains the following lines of data:

*SECTION STIFFNESS, FULL COUPLING
$(K)_{11}$, $(K)_{22}$, $(K)_{33}$, $(K)_{44}$, $(K)_{55}$, $(K)_{66}$, $(K)_{12}$, $(K)_{13}$
$(K)_{14}$, $(K)_{15}$, $(K)_{16}$, $(K)_{23}$, $(K)_{24}$, $(K)_{25}$, $(K)_{26}$, $(K)_{34}$
$(K)_{35}$, $(K)_{36}$, $(K)_{45}$, $(K)_{46}$, $(K)_{56}$
*SECTION INERTIA
(ρA), $(\rho I)_{11}$, $(\rho I)_{12}$, $(\rho I)_{22}$, x_{1cm}, x_{2cm}

In this case, the cross-section properties might represent a fully coupled, linear elastic behavior according to the 21 values of the symmetric 6×6 stiffness matrix. In general, the elastic response to the different beam section strains, transverse shear, axial stretch, bending, and torsion cannot be separated; and no unique centroid and shear center locations can be defined. Depending on the material properties of the two-dimensional cross-section mesh and the options that are defined for the analysis procedure, some coupling terms might vanish, allowing for a transformation into the representation of the cross-section properties that is used with 1-DOF warping elements. Typically, this is the case only if the following conditions are met:

- The beam element centerline has zero curvature and torsion.
- The beam element centerline tangent direction is aligned with the cross-section normal direction.
- All elements have isotropic elasticity or orthotropic elasticity with one of the elasticity tensor principal axes aligned with the cross-section normal direction.

The corresponding properties are then printed into the *jobname*`.dat` data file.

The cross-sectional property information written to the *jobname*`.bsp` text file is read into the general beam section definition in the subsequent beam analysis in Abaqus/Standard as described below.

Contents of the Flexible Body Interface File for 3-DOF Warping Elements

When you use 3-DOF warping elements, a flexible body interface (`.fbi`) file is generated that contains

- the cross-section mesh and its mass and stiffness properties,

- recovery information for reconstructing the in-plane and out-of-plane warping displacements, and
- the associated strain and stress based on beam element cross-section forces and torques.

The `.fbi` file uses a binary format, and you can assign the file to Simpack Nonlinear SIMBEAM element cross-sections. Simpack expects `.fbi` file data in SI units. By default, Abaqus assumes that the cross-section mesh is defined in SI units. You can override the default and specify the units used for the cross-section definition. The data are then converted automatically to SI units before it is written to the `.fbi` file.

Input File Usage: Use both of the following options to specify the units used for the cross-section definition:

**BEAM SECTION GENERATE*
**UNIT SYSTEM*

Using the Generated Cross-Section Properties in a Beam Analysis

As discussed above, the section properties calculated and stored in the *jobname*`.bsp` text file can be used in an actual beam analysis to define cross-sections for beam elements. The data stored in the *jobname*`.bsp` file correspond to the section property data for an arbitrarily shaped solid general beam cross-section meshed with warping elements (see *Defining Linear Section Behavior for Meshed Cross-Sections*). Consequently, a simple method of inserting these data is to include the *jobname*`.bsp` file in the beam analysis. If the *jobname*`.bsp` file is generated using 3-DOF warping elements, you can only use the file in Abaqus/Standard. If the file is generated using 1-DOF warping elements, you can use the file in Abaqus/Standard and Abaqus/Explicit.

Input File Usage: Use the following options to generate section properties in a beam analysis:

**BEAM GENERAL SECTION*, SECTION=MESHED

$\boldsymbol{N_x}$, $\boldsymbol{N_y}$, $\boldsymbol{N_z}$ (direction cosines for $\boldsymbol{n_1}$)
**INCLUDE*, INPUT=*jobname*`.bsp`

Postprocessing from the Beam Model or the Two-Dimensional Cross-Section Model

A tick mark contour plot can be used to visualize stress and strain output along the length of the beam model. All stress and strain components requested for the two-dimensional cross-section model generation are available. Contour plots of stress and strain on the two-dimensional cross-section are also available. The section geometry is read from the output database generated by the two-dimensional cross-section analysis, while the generalized section results are read from the output database generated by the beam analysis.

Initial Conditions

Initial conditions are not meaningful when generating beam section properties and are ignored.

Boundary Conditions

Boundary conditions are not meaningful when generating beam section properties and are ignored.

Loads

Loads are not meaningful when generating beam section properties and are ignored.

Predefined Fields

Temperature and field variables are not allowed for meshed sections.

Material Options

Only the following material behaviors are allowed for meshed sections:

- Isotropic linear elasticity (*Defining Isotropic Elasticity*)
- Orthotropic linear elasticity for 1-DOF warping elements (*Defining Orthotropic Elasticity for 1-DOF Warping Elements*)
- Fully three-dimensional orthotropic linear elasticity for 3-DOF warping elements (*Defining Orthotropic Elasticity by Specifying the Engineering Constants*, *Defining Transversely Isotropic Elasticity*, and *Defining Orthotropic Elasticity by Specifying the Terms in the Elastic Stiffness Matrix*)
- fully anisotropic linear elastic material behavior for 3-DOF warping elements (*Defining Fully Anisotropic Elasticity*)
- density (*Density*)

Elements

Warping elements must be used to mesh the two-dimensional cross-section. See *Warping Elements* for details.

Output

Element output is calculated during the actual beam analysis at the integration points on the meshed cross-section that are selected in the property generation analysis as described above. Output from the property generation analysis is available only on the output database. The Visualization module of Abaqus/CAE can be used to generate contour plots of element output on the cross-section, which requires the output databases from both the section property generation analysis (the cross-section model) and the actual beam analysis. For more information, see the example Python script in *Viewing the analysis of a meshed beam cross-section*.

Input File Template

Generating the Cross-Section Properties in an Abaqus/Standard Analysis for Beams Having Isotropic or Orthotropic, Traction-Type Materials

```
*HEADING
Meshed cross-section
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=WARP2D3, ELSET=TRI
...
*ELEMENT, TYPE=WARP2D4, ELSET=QUAD
...
*SOLID SECTION, MATERIAL=COMPOSITE, ELSET=TRI
*MATERIAL,NAME=COMPOSITE
*ELASTIC, TYPE=TRACTION
E, G1, G2
*DENSITY
...
*SOLID SECTION, MATERIAL=MASS_ONLY, ELSET=QUAD
```

```
*MATERIAL, NAME=MASS_ONLY
*DENSITY
...
*STEP
*BEAM SECTION GENERATE
*SECTION ORIGIN
X, Y
*SECTION POINTS
section point label, element number, integration point number
*END STEP
```

Generating the Cross-Section Properties in an Abaqus/Standard Analysis for Composite Beams Having Fully Three-Dimensional Orthotropic or Fully Anisotropic Materials

```
*HEADING
Meshed cross-section
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=WARPF2D3, ELSET=TRI
...
*ELEMENT, TYPE=WARPF2D4, ELSET=QUAD
...
*ELEMENT, TYPE=WARPF2D6, ELSET=TRI
...
*ELEMENT, TYPE=WARPF2D8, ELSET=QUAD
...
*SOLID SECTION, MATERIAL=ORTHOTROPIC, ELSET=TRI
*MATERIAL,NAME=ORTHOTROPIC
*ELASTIC, TYPE=ENGINEERING CONSTANTS
...
*DENSITY
...
*SOLID SECTION, MATERIAL=ANISOTROPIC, ELSET=QUAD
*MATERIAL, NAME=ANISOTROPIC
...
*DENSITY
...
*STEP
*BEAM SECTION GENERATE
*SECTION ORIGIN
X, Y
*SECTION TANGENT
direction cosine 1, direction cosine 2, direction cosine 3
*SECTION CURVATURE
precurvature 1, precurvature 2, pretorsion
*END STEP
```

Using the Generated Cross-Section Properties in a Subsequent Abaqus/Standard or Abaqus/Explicit Beam Analysis

```
*HEADING
Beam analysis
...
*NODE, NSET=NALL
...
*ELEMENT, TYPE=B31, ELSET=BEAM1
...
*BEAM GENERAL SECTION, SECTION=MESHED
```

N_x, N_y, N_z *(direction cosines for* n_1*)*

```
*INCLUDE, INPUT=jobname.bsp
...
*STEP
*DYNAMIC
...
*BOUNDARY
...
*CLOAD
...
*OUTPUT
*ELEMENT OUTPUT
...
*END STEP
```

Using the Extended Finite Element Method

The extended finite element method (XFEM) allows you to model discontinuities, such as cracks, along an arbitrary, solution-dependent path. The mesh is not required to conform to the discontinued geometry when you study crack growth with XFEM; therefore, you do not need to remesh the bulk materials to account for the presence of the crack.

In this section:

- *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*
- *Applying Cohesive Material Concepts to XFEM-Based Cohesive Behavior*
- *Applying the VCCT Technique to the XFEM-Based LEFM Approach*
- *Contact Interaction of Cracked Element Surfaces*
- *Defining the Constitutive Response of Fluid Flow within the Cracked Element Surfaces*
- *Modeling Slurry Flow*

Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- is commonly referred to as the extended finite element method (XFEM);
- is an extension of the conventional finite element method based on the concept of partition of unity;
- allows the presence of discontinuities in an element by enriching degrees of freedom with special displacement functions;
- does not require the mesh to match the geometry of the discontinuities;
- is a very attractive and effective way to simulate initiation and propagation of a discrete crack along an arbitrary, solution-dependent path without the requirement of remeshing in the bulk materials;
- can be based on traction-separation cohesive behavior or based on the principles of linear elastic fracture mechanics (LEFM);
- can be simultaneously used with the surface-based cohesive behavior approach (see *Contact Cohesive Behavior*) or the Virtual Crack Closure Technique (see *Crack Propagation Analysis*), which are best suited for modeling interfacial delamination;
- can be performed using the static procedure (see *Static Stress Analysis*), the implicit dynamic procedure (see *Implicit Dynamic Analysis Using Direct Integration*), the general fatigue crack growth approach (see *Linear Elastic Fatigue Crack Growth Analysis*), the low-cycle fatigue analysis using the direct cyclic approach (see *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*), the geostatic stress field procedure (see *Geostatic Stress State*), quasi-static analysis (see *Quasi-Static Analysis*), fully coupled thermal-stress analysis (see *Fully Coupled Thermal-Stress Analysis*), or coupled pore fluid diffusion/stress analysis (see *Coupled Pore Fluid Diffusion and Stress Analysis*);
- can also be used to perform contour integral evaluations for an arbitrary stationary surface crack without the need to define the conforming mesh around the crack tip;
- allows the application of distributed pressure loads or distributed heat fluxes to the cracked element surfaces;
- allows the output of some surface variables on the cracked element surfaces;
- allows both material and geometrical nonlinearity; and
- is available only for first-order stress/displacement solid continuum elements, first-order displacement/pore pressure solid continuum elements, first-order displacement/temperature solid continuum elements, first-order displacement/pore pressure/temperature solid continuum elements, and second-order stress/displacement tetrahedral elements.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Modeling Approach

Modeling stationary discontinuities, such as a crack, with the conventional finite element method requires that the mesh conforms to the geometric discontinuities. Creating a conforming mesh can be quite difficult. Modeling a growing crack is even more cumbersome because the mesh must be updated continuously to match the geometry of the discontinuity as the crack progresses.

The extended finite element method (XFEM) alleviates the need to create a conforming mesh. The extended finite element method was first introduced by *Belytschko and Black (1999)*. It is an extension of the conventional finite element method based on the concept of partition of unity by *Melenk and Babuska (1996)*, which allows local enrichment functions to be easily incorporated into a finite element approximation. The presence of discontinuities is ensured by the special enriched functions in conjunction with additional degrees of freedom. However, the finite element framework and its properties such as sparsity and symmetry are retained. XFEM does not alleviate the need for sufficient mesh refinement in the vicinity of the crack tip.

Introducing Nodal Enrichment Functions

For the purpose of fracture analysis, the enrichment functions typically consist of the near-tip asymptotic functions that capture the singularity around the crack tip and a discontinuous function that represents the jump in displacement across the crack surfaces. The approximation for a displacement vector function $\mathbf{u}$ with the partition of unity enrichment is

$$\mathbf{u} = \sum_{I=1}^{N} N_I(x) \left[\mathbf{u}_I + H(x)\,\mathbf{a}_I + \sum_{\alpha=1}^{4} F_\alpha(x)\,\mathbf{b}_I^\alpha \right],$$

where $N_I(x)$ are the usual nodal shape functions; the first term on the right-hand side of the above equation, $\mathbf{u}_I$, is the usual nodal displacement vector associated with the continuous part of the finite element solution; the second term is the product of the nodal enriched degree of freedom vector, $\mathbf{a}_I$, and the associated discontinuous jump function $H(x)$ across the crack surfaces; and the third term is the product of the nodal enriched degree of freedom vector, $\mathbf{b}_I^\alpha$, and the associated elastic asymptotic crack-tip functions, $F_\alpha(x)$. For more details, see *Nodal enrichment functions*.

Accurately modeling the crack-tip singularity requires constantly keeping track of where the crack propagates and is cumbersome because the degree of crack singularity depends on the location of the crack in a nonisotropic material (see *Nodal enrichment functions*). Therefore, we consider the asymptotic singularity functions only when modeling stationary cracks in Abaqus/Standard. Moving cracks are modeled using one of the two alternative approaches described below.

Modeling Moving Cracks with the Cohesive Segments Method and Phantom Nodes

One approach for modeling moving cracks within the framework of XFEM is based on traction-separation cohesive behavior. This approach is used in Abaqus/Standard to simulate both crack initiation and propagation. This is a very general interaction modeling capability, which can be used for modeling brittle or ductile fracture. Other non-XFEM–based approaches for modeling crack initiation and propagation capabilities available in Abaqus/Standard are based either on cohesive elements (*Defining the Constitutive Response of Cohesive Elements Using a Traction-Separation Description*) or on surface-based cohesive behavior (*Contact Cohesive Behavior*). Unlike these non-XFEM–based methods, which require that the cohesive surfaces align with element boundaries and the cracks propagate along a set of predefined paths, the XFEM-based cohesive segments method can be used to simulate both crack initiation and propagation along an arbitrary, solution-dependent path in the bulk materials, since the crack propagation is not tied to the element boundaries in a mesh. In this case the near-tip asymptotic singularity is not needed, and only the displacement jump across a cracked element is considered. Therefore, the crack has to propagate across an entire element at a time to avoid the need to model the stress singularity.

The cohesive segments approach is based on the use of so-called phantom nodes, which are superposed on the original real nodes, and are introduced to represent the discontinuity of the cracked elements (see *Figure 1*). When the element is intact, each phantom node is completely constrained to its corresponding real node. When the element is cut through by a crack, the cracked element splits into two parts. Each part is formed by a combination of some real and phantom nodes depending on the orientation of the crack. Each phantom node and its corresponding real node are no longer tied together and can move apart. For more details, see *Phantom node method*.

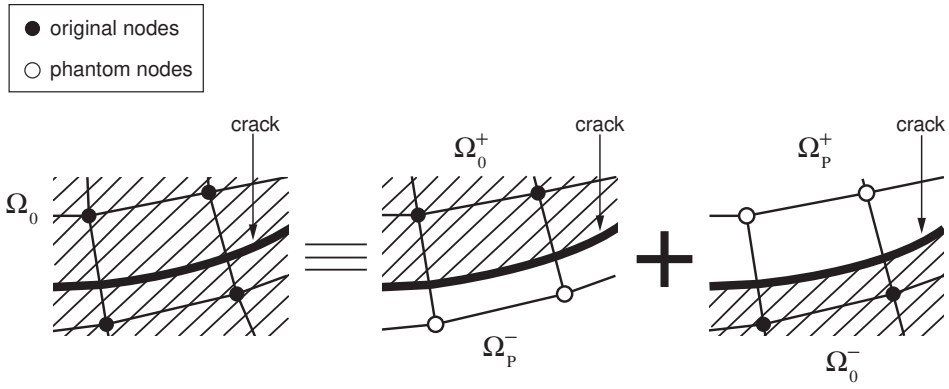


Figure 1: The principle of the phantom node method.

See *Applying Cohesive Material Concepts to XFEM-Based Cohesive Behavior* for more information.

Modeling Hydraulically Driven Fracture

The cohesive segments method in conjunction with phantom nodes discussed above can also be extended to model hydraulically driven fracture. In this case additional phantom nodes with pore pressure degrees of freedom are introduced on the edges of each enriched element to model the fluid flow within the cracked element surfaces in conjunction with the phantom nodes that are superposed on the original real nodes to represent the discontinuities of displacement and fluid pressure in a cracked element. The phantom node at each element edge is not activated until the edge is intersected by a crack. The flow patterns of the pore fluid in the cracked elements are shown in *Figure 2*. The fluid is assumed to be incompressible. The fluid flow continuity, which accounts for both tangential and normal flow within and across the cracked element surfaces as well as the rate of opening of the cracked element surfaces, is maintained. The fluid pressure on the cracked element surfaces contributes to the traction-separation behavior of the cohesive segments in the enriched elements, which enables the modeling of hydraulically driven fracture.

Optionally, phantom nodes with pore pressure, temperature, and slurry concentration degrees of freedom can be introduced on the edges of each enriched element, and phantom nodes with displacement, pore pressure, and temperature degrees of freedom can be superposed on the original real nodes. The temperature degrees of freedom are necessary to model heat transfer due to thermal conduction and radiation across the cracked element surfaces, as well as thermal convection within the cracked element surfaces. The slurry concentration degrees of freedom are necessary to model proppant transport and placement within the cracked surfaces. The temperature and/or slurry degrees of freedom will be activated automatically if thermal and/or slurry material properties are specified for the fluid flow within the cracked element surfaces.

Multiple fluid injections can also be performed in a single analysis.

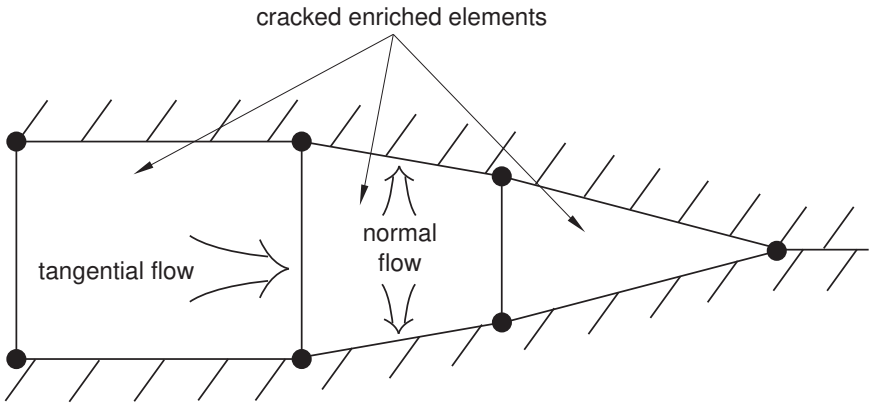


Figure 2: Flow within a cracked element.

See *Defining the Constitutive Response of Fluid Flow within the Cracked Element Surfaces* and *Modeling Slurry Flow* for more information.

Modeling Moving Cracks Based on the Principles of Linear Elastic Fracture Mechanics (LEFM) and Phantom Nodes

An alternative approach to modeling moving cracks within the framework of XFEM is based on the principles of linear elastic fracture mechanics (LEFM). Therefore, it is more appropriate for problems in which brittle crack propagation occurs either on reaching a critical value of a fracture criterion or associated with fatigue crack growth; for more discussion of fracture criteria and fatigue crack growth using the Paris law, see *Fatigue Crack Growth Criterion*, *Linear Elastic Fatigue Crack Growth Analysis*, and *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*. The XFEM-based LEFM approach can be used to simulate crack propagation along an arbitrary, solution-dependent path in the bulk material.

Similar to the XFEM-based cohesive segments method described above, the near-tip asymptotic singularity is not considered, and only the displacement jump across a cracked element is considered. Therefore, the crack has to propagate across an entire element at a time to avoid the need to model the stress singularity. The strain energy release rate at the crack tip is calculated based on the modified Virtual Crack Closure Technique (VCCT), which has been used to model delamination along a known and partially bonded surface (see *Crack Propagation Analysis*).

Another similarity is the introduction of phantom nodes to represent the discontinuity of the cracked element when the fracture criterion is satisfied (for more details. see *Phantom node method*). The real node and the corresponding phantom node will separate when the equivalent strain energy release rate exceeds the critical strain energy release rate at the crack tip in an enriched element. The traction is initially carried as equal and opposite forces on the two surfaces of the cracked element. The traction is ramped down linearly over the separation between the two surfaces with the dissipated strain energy equal to either the critical strain energy required to initiate the separation or the critical strain energy required to propagate the crack depending on whether the VCCT or the enhanced VCCT criterion is specified.

See *Applying the VCCT Technique to the XFEM-Based LEFM Approach* for more information.

Using the Level Set Method to Describe Discontinuous Geometry

A key development that facilitates treatment of cracks in an extended finite element analysis is the description of crack geometry, because the mesh is not required to conform to the crack geometry. The level set method, which is a powerful numerical technique for analyzing and computing interface motion, fits naturally with the extended finite element method and makes it possible to model arbitrary crack growth without remeshing. The crack geometry is defined by two almost-orthogonal signed distance functions, ϕ and ψ, as illustrated in *Figure 3*. For more details, see *Level set method*.

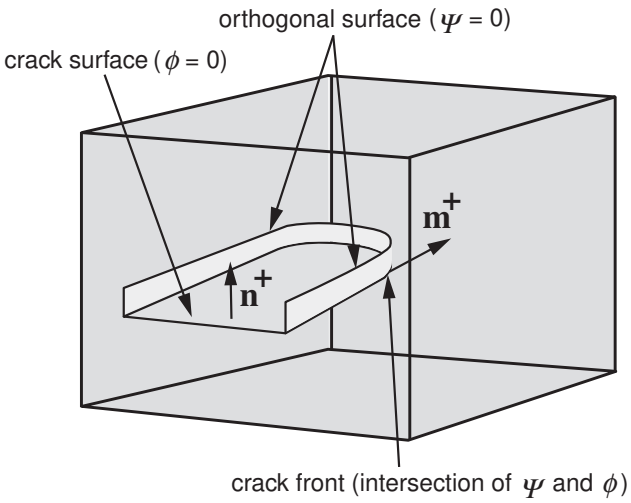


Figure 3: Representation of a nonplanar crack in three dimensions by two signed distance functions ϕ and ψ.

Defining an Enriched Feature and Its Properties

An enriched feature is a region of a model, specified with an element set, in which the finite element interpolation functions are enhanced to include discontinuities. In other words, an enriched feature is a region in the model where you can use the extended finite element method (XFEM) to model a crack. You must specify an enriched feature and its properties to use XFEM. Because there is a computational cost associated with modeling discontinuities, limiting the extent of an enriched feature can enhance computational performance.

An enriched feature can model a stationary crack or a propagating crack, but not both. You must decide during model setup whether a particular enriched feature models a stationary crack or a propagating crack. For a stationary crack, the enrichment functions include the asymptotic elastic crack-tip fields (see *Introducing Nodal Enrichment Functions*). If you model a propagating crack, you must ensure that the enriched feature is large enough to include all areas of the part where the crack could potentially propagate. You should not include areas where the crack is unlikely to propagate in the enriched feature (if this information is known ahead of time). Including these areas increases the computational cost without having any impact on the accuracy of the model. Additional details regarding the criteria for initiation and propagation of a crack are discussed in *Crack Initiation and Direction of Crack Extension*.

The default XFEM implementation in Abaqus does not support interactions among multiple cracks or branching of a single crack. It is best suited for problems where only a single crack is modeled within an enriched feature. Real applications often require modeling multiple cracks within a single part. A part can contain no preexisting cracks or one or more preexisting cracks; it can also contain one or more cracks that initiate during the analysis. For such applications, you can use multiple enriched features within a part (one for each crack) or use a single enriched feature to model more than one crack, although the latter approach has limitations.

The choice of one approach versus the other depends on the ease of specifying multiple enrichment features at the time of model setup. This, in turn, depends on the proximity of the cracks to one another at initiation and during growth. You can choose a single enriched feature in situations where you do not have any prior knowledge regarding the potential locations of multiple cracks or when it is not straightforward to separate out two or more distinct enriched features in a complex part. The following paragraphs provide some guidelines to help you judge what types of situations you can model and to select the correct modeling approach for your problem, starting with the important concept of a *level set conflict*.

As discussed in *Using the Level Set Method to Describe Discontinuous Geometry*, an XFEM crack is represented in terms of nodal values of a level set function. A requirement of the level set representation is that the level set function at any given node must have a unique value. This requirement is violated when more than one crack cuts through a single element or adjacent elements of a given node, resulting in an attempt to assign multiple values of the level set function at the node. Such a condition is referred to as a *level set conflict*. If a level set conflict occurs during an analysis, the analysis ends with an error. Without any kind of prior experimental data, paths of propagating cracks are often difficult to predict. This can cause uncertainty prior to a simulation as to whether a level set conflict will occur if you model multiple cracks in a single enriched feature.

If there are multiple preexisting cracks in a part, you can define them within a single enriched feature as long as these cracks are relatively far away from one another to begin with and do not come close to one another during the crack growth process. This restriction ensures that a level set conflict does not occur at any node. Mesh refinement can help avoid a level set conflict.

By default, crack initiation exhibits the following behavior to avoid level set conflicts:

- Crack initiation in an enriched feature cannot occur until all existing cracks propagate through the enriched feature boundary. As illustrated in *Figure 4*, Crack 2 does not initiate until Crack 1 propagates through the enriched feature boundary.
- Crack initiation cannot occur near a preexisting crack (because this immediately causes a level set conflict).
- More than one crack can initiate at nonpredetermined locations within an enriched feature only if two or more nonadjacent elements satisfy the damage initiation criterion in the same time increment and one of the following conditions is satisfied:

  - There are no preexisting cracks in the enriched feature.
  - All preexisting cracks in the enriched feature have propagated through the feature boundary.

  A level set conflict occurs if the elements in which cracks initiate share nodes.
- A newly initiated second crack cannot approach or enter an already cracked element.

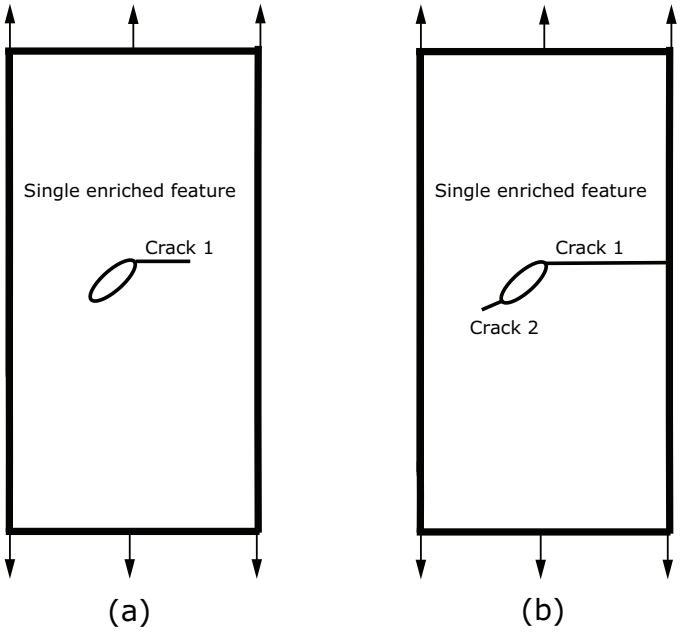


Figure 4: A new crack cannot initiate until the existing crack propagates through the feature boundary.

You can use one of the following methods to change the default behavior described above:

- Use multiple enriched features.

- Use a single enriched feature that is capable of initiating more than one crack at different locations and at different time increments before existing cracks propagate through the feature boundary.

The sections that follow outline these two approaches.

Multiple Enriched Features

To use multiple enriched features, you must define the element sets belonging to each feature at the time of model setup. In other words, you must be able to determine the boundaries among multiple enriched features at the time of model setup. *Figure 5* shows a part with two holes that result in stress concentrations. For the loading shown, two cracks would initiate and grow near these holes. For this part, it is easy to define multiple enriched features during model setup. If both cracks in *Figure 6* are preexisting cracks and do not come close to each other as they grow, you can also use a single enriched feature. However, with a single enriched feature, performance suffers if most of the enriched feature remains uncracked. From a performance point of view, the better choice might still be to use two separate enriched features in the areas where the cracks grow.

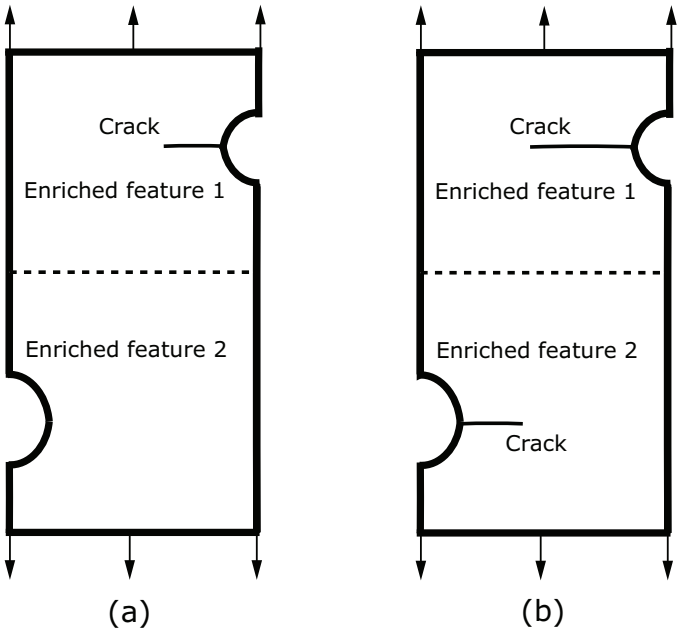


Figure 5: Different stages of crack growth in a part with two enriched features.

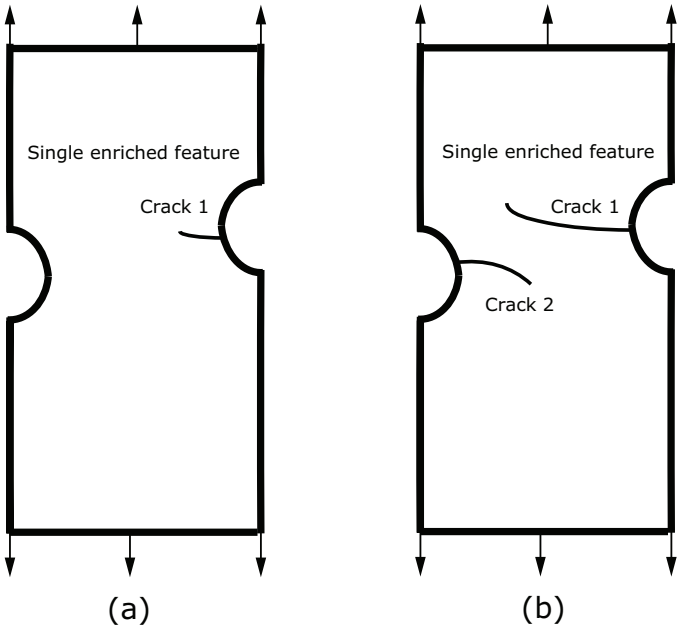


Figure 6: Different stages of crack growth in a part with a single enriched feature.

Single Enriched Feature Capable of Initiating Multiple Cracks before Existing Cracks Propagate through the Feature Boundary

You might not be able to clearly define two or more enriched features at the time of model setup in all situations. *Figure 6*, which shows a plate with two closely spaced holes, illustrates an example of this situation. The potential crack initiation and growth paths are difficult to anticipate ahead of time. It might take some experience and modeling iterations to properly choose the boundaries that separate multiple enriched features. Using a single enriched feature might be more appropriate for this situation. XFEM cannot model cases in which cracks should coalesce during a simulation.

Another example where a clear separation of enriched features is difficult (maybe impossible) involves a composite panel with a hole and with +45/–45 plies. In this panel many parallel matrix cracks could nucleate and propagate in each ply, as illustrated in *Figure 7*. A single enriched feature per ply is more appropriate for this problem.

For the examples shown in *Figure 6* and *Figure 7*, it is recommended that you use a single enriched feature (per ply), but activate the capability to initiate more than one crack at different locations and at different time increments even before preexisting cracks propagate through the feature boundary. This nondefault approach has fewer limitations compared to the default approach discussed earlier. During the model setup phase in this nondefault approach, you explicitly specify that multiple cracks can initiate within a single enriched feature. You also specify a small relative radius, r_s, to define a *crack initiation suppression* zone near a crack tip within which the initiation of another crack is not allowed (to prevent a level set conflict). Multiple cracks are allowed to initiate in an enriched feature, but only outside the crack initiation suppression zone for each individual crack tip (see *Activating and Deactivating the Enriched Feature*). However, a level set conflict could still occur if two cracks propagate close to each other during the crack growth. If this happens, further mesh refinement in the region where the cracks approach each other might help run the analysis to completion.

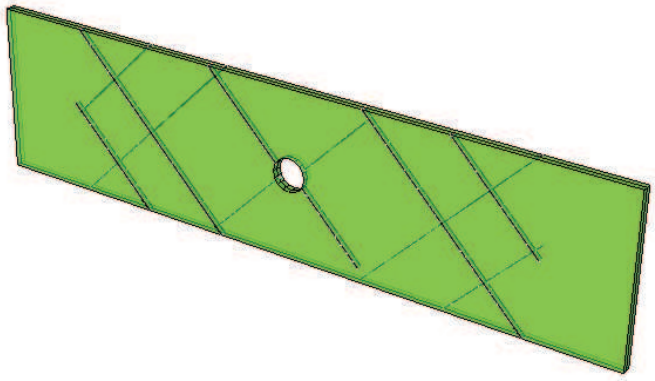


Figure 7: A composite panel with a hole with +45/–45 plies.

Specifying the Enrichment

Abaqus activates the enriched degrees of freedom only when a crack cuts through an element. You can associate only stress/displacement, displacement/pore pressure, displacement/temperature, or displacement/pore pressure/temperature solid continuum elements with an enriched feature.

Input File Usage: **ENRICHMENT*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create->XFEM**

Defining the Type of Enrichment

You can choose to model an arbitrary stationary crack or a discrete crack propagation along an arbitrary, solution-dependent path. The former requires that the elements around the crack tips are enriched with asymptotic functions to catch the singularity and that the elements intersected by the crack interior are enriched with the jump function across the crack surfaces. The latter infers that crack propagation is modeled with either the cohesive segments method or the linear elastic fracture mechanics approach in conjunction with phantom nodes. However, the options are mutually exclusive and cannot be specified simultaneously in a model.

Input File Usage: Use the following option to specify a crack propagation analysis (default):

**ENRICHMENT*, TYPE=PROPAGATION CRACK

Use the following option to specify an analysis with stationary cracks:

**ENRICHMENT*, TYPE=STATIONARY CRACK

Abaqus/CAE Usage: Use the following input to specify a crack propagation analysis:

Interaction module: crack editor: toggle on **Allow crack growth**

Use the following input to specify an analysis with stationary cracks:

Interaction module: crack editor: toggle off **Allow crack growth**

Assigning a Name to the Enriched Feature

You must assign a name to an enriched feature, such as a crack. This name can be used in defining the initial location of the crack surfaces, in identifying a crack for contour integral output, in activating or deactivating the crack propagation analysis, and in generating cracked element surfaces.

Input File Usage: **ENRICHMENT*, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **XFEM**: **Name**: *name*

Identifying an Enriched Region

You must associate the enrichment definition with a region of your model. Only degrees of freedom in elements within these regions are potentially enriched with special functions. The region should consist of elements that are presently intersected by cracks and those that are likely to be intersected by cracks as the cracks propagate.

Input File Usage: **ENRICHMENT*, ELSET=*element set name*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **XFEM**: **Select the crack domain**: select region

Defining a Crack Surface

As a crack propagates through the model, a crack surface representing both facets of cracked elements is generated on those enriched elements that are intersected by a crack during the analysis. You must associate the name of an enriched feature with the surface (see *Assigning a Name to the Enriched Feature* above).

Input File Usage: **SURFACE*, TYPE=XFEM

Abaqus/CAE Usage: An XFEM-based crack surface is not supported in Abaqus/CAE.

Defining Contact Interaction of Enriched Cracked Element Surfaces

Contact interaction of cracked element surfaces, including thermal interaction and pore fluid interaction, is allowed based on a small-sliding formulation or on a finite-sliding formulation within the general contact framework.

See *Contact Interaction of Cracked Element Surfaces* for more information.

Applying Distributed Pressure Loads and Heat Fluxes to Cracked Element Surfaces

When an element is cut by a crack during the analysis, a XFEM-based crack surface is generated during the analysis (see *#unique_2598/unique_2598_Connect_42_simaanl-c-enrichment-surf*). You can apply a distributed pressure load or heat flux to the cracked element surfaces.

Input File Usage: Use the following option to apply a distributed pressure load to a crack surface:

**DSLOAD*
surface name, P or PNU, *magnitude*

Use the following option to apply a distributed heat flux to a crack surface:

**DSFLUX*
surface name, S or SNU, *magnitude*

Abaqus/CAE Usage: An XFEM-based crack surface is not supported in Abaqus/CAE.

Specifying the Initial Location of an Enriched Feature

Because the mesh is not required to conform to the geometric discontinuities, the initial location of a preexisting crack must be specified in the model. The level set method is provided for this purpose. Two signed distance functions per node are generally required to describe a crack geometry. The first describes the crack surface, while the second is used to construct an orthogonal surface so that the intersection of the two surfaces gives the crack front (see *Initial Conditions*).

The first signed distance function must be either greater or less than zero and cannot be equal to zero. If an initial crack has to be defined at the boundaries of an element, a very small positive or negative value for the first signed distance function must be specified.

Input File Usage: Use the following option to specify the initial location of an enriched feature:

**INITIAL CONDITIONS*, TYPE=ENRICHMENT

Abaqus/CAE Usage: Interaction module: crack editor: **Crack location**: **Select**: select region

Activating and Deactivating the Enriched Feature

You can activate or deactivate the crack propagation capability within the step definition.

You can specify a small relative radius, r_s, around the crack tip (as shown in *Figure 8*) to define a crack initiation suppression zone within which the elements in the enriched feature are excluded from crack nucleation. However, multiple crack nucleations are allowed to occur when the damage initiation criterion is satisfied in the elements lying outside the crack initiation suppression zone in the enriched feature. The default radius value is five times the typical element characteristic length in the enriched region.

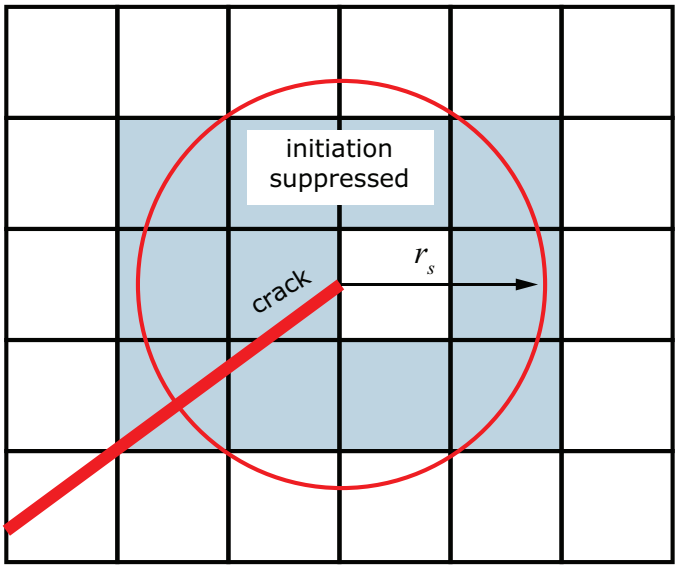


Figure 8: Crack initiation suppression zone.

Input File Usage: Use the following option to activate the crack propagation capability within the step definition:

**ENRICHMENT ACTIVATION*, NAME=*name*, ACTIVATE=ON (default)

Use the following option to deactivate the crack propagation capability within the step definition:

**ENRICHMENT ACTIVATION*, NAME=*name*, ACTIVATE=OFF

Use the following option to deactivate the crack propagation capability automatically once all the preexisting cracks (or if there are no preexisting cracks, all the allowable newly nucleated cracks) have propagated through the boundary of the given enriched feature within the step definition:

**ENRICHMENT ACTIVATION*, NAME=*name*, ACTIVATE=AUTO OFF

Use the following option to activate the crack propagation capability for a given enriched feature with the possibility of multiple crack nucleations within the step definition:

**ENRICHMENT ACTIVATION*, NAME=*name*, ACTIVATE=MULTICRACKS, CRACK INITIATION RADIUS=r_s

Abaqus/CAE Usage: To modify the status of the crack propagation capability in a step, you must first create an **XFEM crack growth** interaction:

Interaction module: **Create Interaction**: select initial step: **XFEM Crack Growth**: select crack: **Interaction manager**: select interaction in step: **Edit**: toggle on/off **Allow crack growth in this step**

Activating the crack propagation capability for a given enriched feature with the possibility of multiple crack nucleations is not supported in Abaqus/CAE.

Contour Integral

When you evaluate the contour integrals using the conventional finite element method (*Contour Integral Evaluation*), you must define the crack front explicitly and specify the virtual crack extension direction in addition to matching the mesh to the cracked geometry. Detailed focused meshes are generally required and obtaining accurate contour integral results for a crack in a three-dimensional curved surface can be cumbersome. The extended finite element in conjunction with the level set method alleviates these shortcomings. The adequate singular asymptotic fields and the discontinuity are ensured by the special enrichment functions in conjunction with additional degrees of freedom. In addition, the crack front and the virtual crack extension direction are determined automatically by the level set signed distance functions.

Input File Usage: Use one of the following options to obtain contour integrals for a named enriched feature with the extended finite element method:

**CONTOUR INTEGRAL*, XFEM=DOMAIN, CRACK NAME=*name*

**CONTOUR INTEGRAL*, XFEM=LINE (default), CRACK NAME=*name*

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*

Specifying the Enrichment Radius

Although XFEM has alleviated the shortcomings associated with defining the conforming mesh in the neighborhood of the crack front, you must still generate a sufficient number of elements around the crack front to obtain path-independent contours, particularly in a region with high crack-front curvature. The group of elements within a small radius from the crack front are enriched and become involved in the contour integral calculations. The default enrichment radius is six times the typical element characteristic length of those elements along the crack front in the enriched area. You must include the elements inside the enrichment radius in the element set used to define the enriched region. Elements with a large aspect ratio should be avoided along the crack front.

Input File Usage: Use the following option to specify an enrichment radius:

**ENRICHMENT*, ENRICHMENT RADIUS

Abaqus/CAE Usage: Interaction module: crack editor: **Enrichment radius**: **Analysis default** or **Specify**

Adjusting the Node Spacing and Radius

By default, Abaqus will adjust the locations of the nodes along a three-dimensional XFEM crack and make the node spacing equal, and then modify the corresponding radii used for contour integral based on the local line curvature. You can deactivate these adjustments for the line integral calculation by using the following specification.

Input File Usage: Use the following option to deactivate the adjustments of nodal spacing and radius:

**CONTOUR INTEGRAL*, LENGTH SCALING FACTOR

Abaqus/CAE Usage: Deactiving the adjustments of nodal spacing and radius is not supported in Abaqus/CAE.

Procedures

Modeling discontinuities as an enriched feature can be performed using any of the following:

- static analysis (see *Static Stress Analysis*);
- implicit dynamic analysis (see *Implicit Dynamic Analysis Using Direct Integration*);
- low-cycle fatigue analysis using the direct cyclic approach (*Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*);
- fatigue crack growth analysis (see *Linear Elastic Fatigue Crack Growth Analysis*);
- fully coupled thermal-stress analysis (see *Fully Coupled Thermal-Stress Analysis*);
- quasi-static analysis (see *Quasi-Static Analysis*);
- geostatic stress field analysis (see *Geostatic Stress State*); or
- coupled pore fluid diffusion/stress analysis (see *Coupled Pore Fluid Diffusion and Stress Analysis*).

Initial Conditions

Initial conditions to identify initial boundaries or interfaces of an enriched feature can be specified (see *Initial Conditions*).

Boundary Conditions

Boundary conditions can be applied to any of the displacement, temperature, or pore pressure degrees of freedom (see *Boundary Conditions*).

Loads

The following types of loading can be prescribed in a model with an enriched feature:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–3) or the pore pressure degree of freedom (8); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.
- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Node-based film and radiation conditions.
- Average-temperature radiation conditions.
- Element- and surface-based film and radiation conditions.

For more information on heat fluxes, film conditions, and radiation conditions, see *Thermal Loads*.

Predefined Fields

The following predefined fields can be specified in a model with an enriched feature, as described in *Predefined Fields*:

- Nodal temperatures (although temperature is not a degree of freedom in stress/displacement elements). The specified temperature affects temperature-dependent critical stress and strain failure criteria.

- The values of user-defined field variables. The specified value affects field-variable-dependent material properties.

Material Options

Any of the mechanical constitutive models in Abaqus/Standard, including user-defined materials (defined using user subroutine *UMAT*) can be used to model the mechanical behavior of the enriched element in a crack propagation analysis. See *Abaqus Materials Guide*. The inelastic definition at a material point must be used in conjunction with the linear elastic material model (*Linear Elastic Behavior*), the porous elastic material model (*Elastic Behavior of Porous Materials*), or the hypoelastic material model (*Hypoelastic Behavior*). Only isotropic elastic materials are supported when evaluating the contour integral for a stationary crack.

Elements

Only the following elements can be associated with an enriched feature:

- First-order solid continuum stress/displacement elements.
- First-order displacement/pore pressure solid continuum elements.
- First-order displacement/temperature solid continuum elements.
- First-order displacement/pore pressure/temperature solid continuum elements.
- Second-order stress/displacement tetrahedral elements.

For propagating cracks these include

- bilinear plane strain and plane stress elements,
- bilinear axisymmetric elements,
- linear brick elements,
- linear tetrahedral elements, and
- second-order tetrahedral elements.

For stationary cracks, these include

- linear brick elements,
- linear tetrahedral elements, and
- second-order tetrahedral elements.

For an incompatible mode element, Abaqus/Standard discards the contribution due to the incompatible deformation mode immediately after the element is fractured under a tensile loading. Therefore, the stress level at the cracked element may not return completely to its originally unloaded state even when this cracked element is unloaded completely and the contact of the cracked element surfaces is reestablished.

Output

In addition to the standard output identifiers available in Abaqus (*Abaqus/Standard Output Variable Identifiers*), the following nodal, whole element, and surface variables have special meaning for a model with an enriched feature.

Nodal variables:

PHILSM	Signed distance function to describe the crack surface.
PSILSM	Signed distance function to describe the initial crack front.

Whole element variables:

STATUSXFEM Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked and 0.0 if the element contains no crack. If the element is partially cracked, the value of STATUSXFEM lies between 1.0 and 0.0.)

LOADSXFEM Distributed pressure loads applied to the crack surface.

Surface variables (available only for propagating cracks modeled with first-order solid continuum elements):

CRKDISP Crack opening and relative tangential motions on cracked surfaces in enriched elements.

CSDMG Damage variable on cracked surfaces in enriched elements.

CRKSTRESS Remaining residual pressure and tangential shear stresses on cracked surfaces in enriched elements.

Use of Unsymmetric Matrix Storage and Solution

When the pore pressure degrees of freedom are activated in the enriched elements, matrices are unsymmetric; therefore, unsymmetric matrix storage and solution may be needed to improve convergence (see *Matrix Storage and Solution Scheme in Abaqus/Standard*).

Visualization

A crack can be visualized through the iso-surface for the signed distance function PHILSM.

If a crack cuts through a very tiny corner of an enriched element, the displacements along the crack front in the enriched element may be distorted in rare cases in the Visualization module of Abaqus/CAE (Abaqus/Viewer) when displaying the contours. The distortion, however, is not present when viewing only the deformed shape.

When an element is cut through by a crack, the cracked element splits into two parts, each part formed by a real domain and a phantom domain, as illustrated in *Figure 1*. Contour plot integration point values for cracked elements consider contributions from only the real domains in both parts of the cracked elements. However, when you probe cracked elements, only the contribution from the part of the elements containing the real domain Ω_0^- and the phantom domain Ω_p^+ is reported.

When evaluating the contour integrals in a stationary crack, additional integration stations are introduced internally in the elements enriched with singular asymptotic crack-tip fields. However, visualization of the element output variables in those additional integration points is not supported in the Visualization module of Abaqus/CAE (Abaqus/Viewer).

Limitations

The following limitations exist with an enriched feature:

- An enriched element cannot be intersected by more than one crack.
- A crack is not allowed to turn more than 90° in one increment during an analysis.
- Only asymptotic crack-tip fields in an isotropic elastic material are considered for a stationary crack.
- Adaptive remeshing is not supported.
- Composite solid elements are not supported.

- Import analysis is not supported.

Input File Template

The following is an example of modeling crack propagation with the XFEM-based cohesive segments method:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=PROPAGATION CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, INTERACTION=INTERACTION
*SURFACE, TYPE=XFEM, NAME=SURF_NAME
Data lines to specify the names of enriched features
*MATERIAL, NAME=STEEL1
...
*MATERIAL, NAME=STEEL2
*DAMAGE INITIATION, CRITERION=MAXPS, TOLERANCE=0.05
*DAMAGE EVOLUTION, TYPE=ENERGY
Data lines to specify the failure mechanism
...
*SURFACE INTERACTION, NAME=INTERACTION
*SURFACE BEHAVIOR
Data lines to specify the contact of cracked element surfaces
...
*STEP
*STATIC
...
*END STEP
*STEP
*STATIC
...

*ENRICHMENT ACTIVATION, TYPE=PROPAGATION CRACK,
NAME=ENRICHMENT, ACTIVATE=OFF
...
*END STEP
```

The following is an example of modeling crack propagation with the XFEM-based LEFM approach:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=PROPAGATION CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, INTERACTION=INTERACTION
*MATERIAL, NAME=STEEL1
...
*MATERIAL, NAME=STEEL2
```

```
*DAMAGE INITIATION, CRITERION=MAXPS, TOLERANCE=0.05
Data lines to specify the crack nucleation mechanism
...
*SURFACE INTERACTION, NAME=INTERACTION
*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=VCCT, TOLERANCE=0.05,VISCOSITY=0.00001
Data lines to specify the crack propagation criterion
...
*END STEP
```

The following is an example of calculating contour integrals in stationary cracks with the extended finite element method:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=STATIONARY CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, ENRICHMENT RADIUS
*MATERIAL, NAME=STEEL1
...
*MATERIAL, NAME=STEEL2
...
*STEP
*STATIC
...
*CONTOUR INTEGRAL, CRACK NAME=ENRICHMENT, XFEM
*END STEP
```

References

- Belytschko, T., and T. Black, “Elastic Crack Growth in Finite Elements with Minimal Remeshing,” *International Journal for Numerical Methods in Engineering*, vol. 45, pp. 601–620, 1999.
- Melenk, J., and I. Babuska, “The Partition of Unity Finite Element Method: Basic Theory and Applications,” *Computer Methods in Applied Mechanics and Engineering*, vol. 39, pp. 289–314, 1996.

Applying Cohesive Material Concepts to XFEM-Based Cohesive Behavior

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- can be based on traction-separation cohesive behavior;
- can be used in Abaqus/Standard to simulate both crack initiation and propagation;
- is a very general interaction modeling capability;
- can be used for modeling brittle or ductile fracture; and
- can be simultaneously used with the surface-based cohesive behavior approach (see *Contact Cohesive Behavior*) or the Virtual Crack Closure Technique (see *Crack Propagation Analysis*), which are best suited for modeling interfacial delamination.

The formulas and laws that govern the behavior of XFEM-based cohesive segments for a crack propagation analysis are very similar to those used for cohesive elements with traction-separation constitutive behavior (*Defining the Constitutive Response of Cohesive Elements Using a Traction-Separation Description*) and those used for surface-based cohesive behavior (*Contact Cohesive Behavior*). The similarities extend to the linear elastic traction-separation model, damage initiation criteria, and damage evolution laws.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- **DAMAGE EVOLUTION*
- **DAMAGE INITIATION*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Linear Elastic Traction-Separation Behavior

The available traction-separation model in Abaqus assumes initially linear elastic behavior followed by the initiation and evolution of damage. The elastic behavior is written in terms of an elastic constitutive matrix that relates the normal and shear stresses to the normal and shear separations of a cracked element.

The nominal traction stress vector, $\mathbf{t}$, consists of the following components: t_n, t_s, and (in three-dimensional problems) t_t, which represent the normal and the two shear tractions, respectively. The corresponding separations are denoted by δ_n, δ_s, and δ_t. The elastic behavior can then be written as

$$\mathbf{t} = \begin{Bmatrix} t_n \\ t_s \\ t_t \end{Bmatrix} = \begin{bmatrix} K_{nn} & 0 & 0 \\ 0 & K_{ss} & 0 \\ 0 & 0 & K_{tt} \end{bmatrix} \begin{Bmatrix} \delta_n \\ \delta_s \\ \delta_t \end{Bmatrix} = \mathbf{K}\boldsymbol{\delta}.$$

The normal and tangential stiffness components will not be coupled: pure normal separation by itself does not give rise to cohesive forces in the shear directions, and pure shear slip with zero normal separation does not give rise to any cohesive forces in the normal direction.

The terms K_{nn}, K_{ss}, and K_{tt} are calculated based on the elastic properties for an enriched element. Specifying the elastic properties of the material in an enriched region is sufficient to define both the elastic stiffness and the traction-separation behavior. For simplicity, we assume that $K_{nn} = K_{ss} = K_{tt}$.

Input File Usage: Alternatively, the elastic stiffness can be specified directly by using one of the following options:

**SURFACE BEHAVIOR*, PRESSURE-OVERCLOSURE=LINEAR
**SURFACE BEHAVIOR*, PENALTY
**SURFACE BEHAVIOR*, AUGMENTED LAGRANGE

Specifically, if a user-defined material is used, you must define the elastic stiffness using one of the options above.

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical->Normal Behavior**: **Constraint enforcement method: Default: Pressure-Overclosure: Linear**; **Penalty (Standard)**; or **Augmented Lagrange (Standard)**

Damage Modeling

Damage modeling allows you to simulate the degradation and eventual failure of an enriched element. The failure mechanism consists of two ingredients: a damage initiation criterion and a damage evolution law. The initial response is assumed to be linear as discussed in the previous section. However, once a damage initiation criterion is met, damage can occur according to a user-defined damage evolution law. *Figure 1* shows a typical linear and a typical nonlinear traction-separation response with a failure mechanism. The enriched elements do not undergo damage under pure compression.

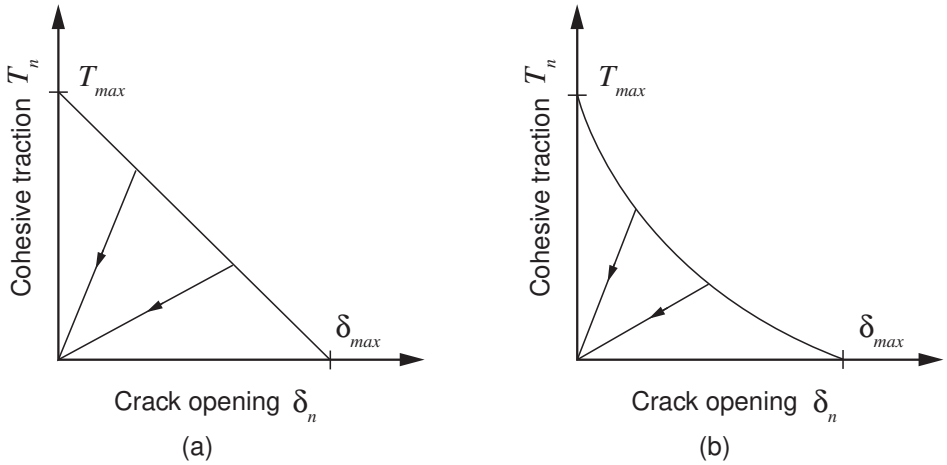


Figure 1: Typical linear (a) and nonlinear (b) traction-separation response.

Damage of the traction-separation response for cohesive behavior in an enriched element is defined within the same general framework used for conventional materials (see *About Progressive Damage and Failure*). However, unlike cohesive elements with traction-separation behavior, you do not have to specify the undamaged traction-separation behavior in an enriched element.

Crack Initiation and Direction of Crack Extension

Crack initiation refers to the beginning of degradation of the cohesive response at an enriched element. The process of degradation begins when the stresses or the strains satisfy specified crack initiation criteria. Crack initiation criteria are available based on the following Abaqus/Standard built-in models:

- the maximum principal stress criterion,
- the maximum principal strain criterion,
- the maximum nominal stress criterion,
- the maximum nominal strain criterion,
- the quadratic traction-interaction criterion,
- the quadratic separation-interaction criterion, and
- the three-dimensional LaRC05 criterion.

In addition, a user-defined damage initiation criterion can be specified in user subroutine *UDMGINI*.

An additional crack is introduced or the crack length of an existing crack is extended after an equilibrium increment when the fracture criterion, f, reaches the value 1.0 within a given tolerance:

$$1.0 \le f \le 1.0 + f_{tol}.$$

You can specify the tolerance f_{tol}. If $f > 1 + f_{tol}$, the time increment is cut back such that the crack initiation criterion is satisfied. The default value of f_{tol} is 0.05. To improve performance, a separate tolerance f_{tol}^{g} can be specified to control the crack growth of an existing crack while f_{tol} is used to control the nucleation of an additional crack. If it is not specified, the growth tolerance, f_{tol}^{g}, is set equal to f_{tol}.

Input File Usage: **DAMAGE INITIATION*, TOLERANCE=f_{tol}, GROWTH TOLERANCE=f_{tol}^{g}

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Tolerance**: f_{tol}

Fracture of Multiple Elements in an Unstable Crack Growth Analysis

For an unstable crack growth problem, sometimes it is more efficient to allow multiple elements at and ahead of a crack tip to fracture without excessively cutting back the increment size when the fracture criterion is satisfied. Abaqus/Standard activates this capability automatically if you specify an unstable growth tolerance, f_{tol}^{u}. In this case if the fracture criterion, f, is within the given unstable growth tolerance:

$$1.0 + f_{tol} \le f \le 1.0 + f_{tol}^{u},$$

where f_{tol} is the tolerance described earlier in this section, Abaqus/Standard immediately reduces the time increment size by default to a very small value, $\beta \triangle t_{\min}$. Reducing the time increment size allows more elements to fracture until $f < 1$ for all the elements ahead of the crack tip. You can, however, optionally specify the maximum number of cutbacks allowed, n, to be controlled by the regular tolerance, f_{tol}, prior to the activation of the unstable growth tolerance in an increment. After this limit the time increment size is recovered automatically to a larger value, $\alpha \triangle t_{pre}$, where:

$\Delta t_{\min}$
Minimum time increment allowed

Δt_{pre}
Time increment size prior to the unstable crack growth

α (default 0.5), β (default 2.0), and n (default 0)
Scaling parameters

If you do not specify a value for the unstable growth tolerance, the default value is infinity. In this case the fracture criterion, *f*, for unstable crack growth is not limited by any upper bound value in the above equation.

Input File Usage: Use both of the following options:

**DAMAGE INITIATION*, UNSTABLE GROWTH TOLERANCE=f^u_{tol}
**CONTROLS*, TYPE=NO CUTBACK SCALING

Abaqus/CAE Usage: Use the following option to specify an unstable crack growth tolerance:

Interaction module: **Interaction**->**Property**->**Create**: **Contact**:**Mechanical**->**Fracture Criterion**, Toggle on **Specify tolerance for unstable crack propagation**, select **Specify value**

Specifying the Crack Direction

When the maximum principal stress or the maximum principal strain criterion is specified, the newly introduced crack is always orthogonal to the maximum principal stress/strain direction when the fracture criterion is satisfied. However, when one of the other Abaqus/Standard built-in crack initiation criteria is used, you have to specify if the newly introduced crack will be orthogonal to the element local 1-direction or orthogonal to the element local 2-direction (see *Conventions*) when the fracture criterion is satisfied. By default, the crack is orthogonal to the element local 1-direction. If a user-defined damage initiation criterion is specified, the normal direction to the crack plane or the crack line can be defined in user subroutine *UDMGINI*.

Input File Usage: Use one of the following options to specify the crack direction when the maximum nominal stress, the maximum nominal strain, the quadratic traction-interaction, or the quadratic separation-interaction criterion is specified:

**DAMAGE INITIATION*, NORMAL DIRECTION=1 (default)

**DAMAGE INITIATION*, NORMAL DIRECTION=2

Abaqus/CAE Usage: Property module: material editor: **Mechanical**->**Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, or **Maxs Damage**: **Direction relative to local 1-direction (for XFEM)**: **Normal** or **Parallel**

Maximum Principal Stress Criterion

The maximum principal stress criterion can be represented as

$$\mathbf{f} = \left\{ \frac{\langle \sigma_{max} \rangle}{\sigma^o_{max}} \right\}.$$

Here, σ^o_{max} represents the maximum allowable principal stress. The symbol $\langle\rangle$ represents the Macaulay bracket with the usual interpretation (that is, $\langle\sigma_{max}\rangle = 0$ if $\sigma_{max} < 0$ and $\langle\sigma_{max}\rangle = \sigma_{max}$ if $\sigma_{max} \geq 0$). The Macaulay brackets are used to signify that a purely compressive stress state does not initiate damage. Damage is assumed to initiate when the maximum principal stress ratio (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=MAXPS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxps Damage**

Maximum Principal Strain Criterion

The maximum principal strain criterion can be represented as

$$\mathrm{f} = \left\{ \frac{\langle \varepsilon_{max} \rangle}{\varepsilon^o_{max}} \right\}.$$

Here, ε^o_{max} represents the maximum allowable principal strain, and the Macaulay brackets signify that a purely compressive strain does not initiate damage. Damage is assumed to initiate when the maximum principal strain ratio (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=MAXPE

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxpe Damage**

Maximum Nominal Stress Criterion

The maximum nominal stress criterion can be represented as

$$\mathrm{f} = \max\left\{ \frac{\langle t_n \rangle}{t^o_n}, \frac{t_s}{t^o_s}, \frac{t_t}{t^o_t} \right\}.$$

The nominal traction stress vector, $\mathbf{t}$, consists of three components (two in two-dimensional problems). t_n is the component normal to the likely cracked surface, and t_s and t_t are the two shear components on the likely cracked surface. Depending on what you specify (see *Specifying the Crack Direction* above), the likely cracked surface will be orthogonal either to the element local 1-direction or to the element local 2-direction. Here, t^o_n, t^o_s, and t^o_t represent the peak values of the nominal stress. The symbol $\langle\rangle$ represents the Macaulay bracket with the usual interpretation. The Macaulay brackets are used to signify that a purely compressive stress state does not initiate damage. Damage is assumed to initiate when the maximum nominal stress ratio (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=MAXS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxs Damage**

Maximum Nominal Strain Criterion

The maximum nominal strain criterion can be represented as

$$\mathrm{f} = \max\left\{\frac{\langle \varepsilon_n \rangle}{\varepsilon_n^o}, \frac{\varepsilon_s}{\varepsilon_s^o}, \frac{\varepsilon_t}{\varepsilon_t^o}\right\}.$$

Damage is assumed to initiate when the maximum nominal strain ratio (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=MAXE

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxe Damage**

Quadratic Nominal Stress Criterion

The quadratic nominal stress criterion can be represented as

$$\mathrm{f} = \left\{\frac{\langle t_n \rangle}{t_n^o}\right\}^2 + \left\{\frac{t_s}{t_s^o}\right\}^2 + \left\{\frac{t_t}{t_t^o}\right\}^2.$$

Damage is assumed to initiate when the quadratic interaction function involving the stress ratios (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=QUADS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quads Damage**

Quadratic Nominal Strain Criterion

The quadratic nominal strain criterion can be represented as

$$\mathrm{f} = \left\{\frac{\langle \varepsilon_n \rangle}{\varepsilon_n^o}\right\}^2 + \left\{\frac{\varepsilon_s}{\varepsilon_s^o}\right\}^2 + \left\{\frac{\varepsilon_t}{\varepsilon_t^o}\right\}^2.$$

Damage is assumed to initiate when the quadratic interaction function involving the nominal strain ratios (as defined in the expression above) reaches a value of one.

Input File Usage: **DAMAGE INITIATION*, CRITERION=QUADE

Abaqus/CAE Usage: Property module: material editor: : **Damage for Traction Separation Laws**: **Quade Damage**

LaRC05 Three-Dimensional Criterion

The LaRC05 three-dimensional criterion can be applied generally to polymer-matrix fiber-reinforced composites. This criterion considers four different damage initiation mechanisms: matrix cracking, fiber kinking, fiber splitting, and fiber tension. For detailed information on the damage initiation criterion, see *LaRC05 Criterion*.

The initiation criterion that first reaches a value of 1.0 determines the damage initiation.

Input File Usage: **DAMAGE INITIATION*, CRITERION=LARC05

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Fiber-Reinforced Composites->LaRC05 Damage**

User-Defined Damage Initiation Criterion

User subroutine *UDMGINI* provides a general capability for implementing a user-defined damage initiation criterion.

You can define several damage initiation mechanisms in user subroutine *UDMGINI*. You represent each damage initiation mechanism by a fracture criterion, f_{indexi}, and its associated normal direction to the crack plane or the crack line. Although you can define several damage initiation mechanisms, the actual damage initiation for an enriched element is governed by the most severe damage initiation mechanism:

$$\mathrm{f} = \max\left\{f_{index1}, f_{index2}, \ldots f_{indexn}\right\}.$$

Damage is assumed to initiate when f, as defined in the expression above, reaches a value of one.

You must specify any material constants that are needed in user subroutine *UDMGINI* as part of a user-defined damage initiation criterion definition.

Input File Usage: Use the following option to define a user-defined damage initiation criterion:

**DAMAGE INITIATION*, CRITERION=USER

Use the following option to specify the total number of failure mechanisms in the user-defined damage initiation criterion:

**DAMAGE INITIATION*, CRITERION=USER, FAILURE MECHANISMS=n

Use the following option to define properties for a user-defined damage initiation criterion:

**DAMAGE INITIATION*, CRITERION=USER, PROPERTIES=*number_of_constants*

Abaqus/CAE Usage: Defining a user-defined damage initiation criterion is not supported in Abaqus/CAE.

Limiting the Crack Propagation Direction

When the maximum principal stress, maximum principal strain, or user-defined damage initiation criterion is specified, you can limit the new crack propagation direction to within a certain angle (in degrees) of the previous crack propagation direction. The default is 85°.

Input File Usage: Use the following option to set the maximum allowed change in the crack propagation angle (in degrees):

**DAMAGE INITIATION*, ANGLEMAX=*max_angle*

Abaqus/CAE Usage: Limiting the crack propagation direction is not supported in Abaqus/CAE.

Local Calculations of the Stress and Strain Fields Ahead of the Crack Tip

An accurate and efficient evaluation of the stress/strain fields ahead of the crack tip is important for both evaluating the crack initiation criterion and computing the crack propagation direction when needed. Abaqus/Standard offers several options for computing these fields.

Centroidal Values of Stress and Strain

By default, the stress/strain computed at the element centroid ahead of the crack tip is used to determine if the damage initiation criterion is satisfied and to determine the crack propagation direction. See *Figure 2*.

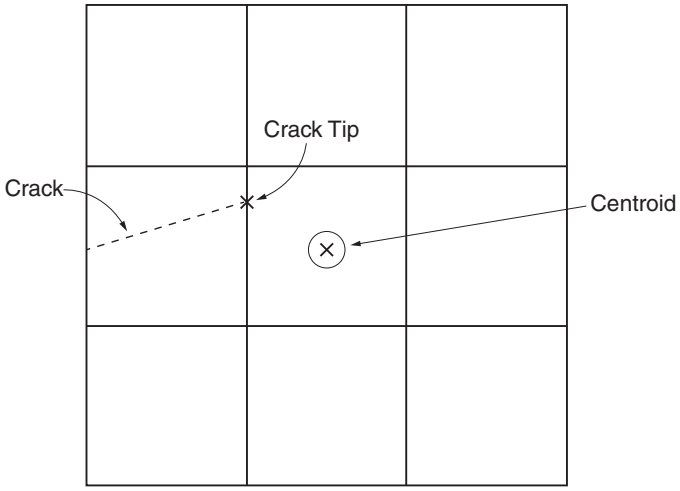


Figure 2: Centroidal and crack tip locations.

Input File Usage: **DAMAGE INITIATION*, POSITION=CENTROID (default)

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position->Centroid**

Computing the Stress and Strain Fields at the Crack Tip

With a sufficiently refined mesh, the centroidal approximation is accurate and economical. However, if the finite element mesh in the vicinity of the crack tip is coarse relative to the gradients in the stress/strain fields, the default centroidal approximation may not be sufficient. In such cases you can use the stress/strain extrapolated to the crack tip to determine if the damage initiation criterion is satisfied and to determine the crack propagation direction. See *Figure 2*.

Input File Usage: **DAMAGE INITIATION*, POSITION=CRACKTIP

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position->Crack tip**

Combining Crack Tip and Centroidal Calculations

You can also choose to combine the two previous alternatives: you can use the stress/strain values extrapolated to the crack tip to determine if the damage initiation criterion is satisfied, and you can use the stress/strain values at the element centroid to determine the crack propagation direction.

Input File Usage: **DAMAGE INITIATION*, POSITION=COMBINED

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position->Combined**

Nonlocal Averaging of the Stress/Strain Fields and Smoothing of the Crack Surface Normals to Improve the Accuracy of Crack Propagation Directions

The three options for evaluating the stress and strain fields discussed above are local calculations in the sense that the evaluated fields are local to the single element ahead of the crack tip. In the case of coarse and/or unstructured meshes a nonlocal averaging of the stress and strain fields ahead of the crack tip can lead to a more accurate evaluation of those fields, which can improve the accuracy of the computed propagation directions. In addition, a moving least-squares approximation by polynomials is used by default to obtain more accurate crack propagation directions. The least-squares approximation further smooths out the normals of the individual crack facets in elements along the crack front that satisfy the damage initiation criterion.

Input File Usage: **DAMAGE INITIATION*, POSITION=NONLOCAL

Abaqus/CAE Usage: Nonlocal averaging of the stress/strain fields and smoothing of the crack surface normals are not supported in Abaqus/CAE.

Specifying the Region of the Model Used for Nonlocal Averaging and Smoothing

To control the range of elements used for nonlocal averaging and smoothing in the crack direction calculations, you can specify a radius, r_c, within which the elements ahead of the crack tip are included (see *Figure 3*). The default radius is three times the typical element characteristic length in the enriched region.

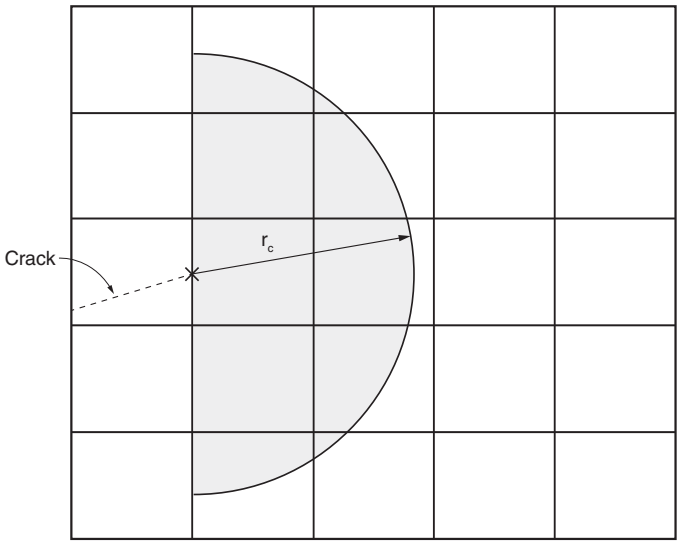


Figure 3: Nonlocal averaging region.

Input File Usage: **DAMAGE INITIATION*, R CRACK DIRECTION=r_c

Abaqus/CAE Usage: Specifying the range of the model for nonlocal averaging and smoothing is not supported in Abaqus/CAE.

Smoothing the Stress/Strain Fields before Averaging

To further improve the nonlocal averaging, you can request an initial smoothing of the stress/strain fields ahead of the crack. In this case Abaqus/Standard averages the field values to element nodes and then interpolates the smoothed fields to the integration points. Once smoothing is complete, the nonlocal averaging is applied. No smoothing is applied by default.

Input File Usage: Use one of the following options:

**DAMAGE INITIATION*, SMOOTHING=NONE (default)
**DAMAGE INITIATION*, SMOOTHING=NODAL

Abaqus/CAE Usage: Smoothing the stress/strain fields before averaging is not supported in Abaqus/CAE.

Weighting Schemes for Nonlocal Averaging

Abaqus/Standard offers a number of weighting schemes for field smoothing that provide additional control over nonlocal averaging. For example, you may want to give a higher weighting to elements close to the crack tip. You can specify a weight function, ω, to compute the average stress/strain based on the distance from the element integration points to the crack tip, r. By default, a uniform weighting is applied to all elements used for averaging; alternatively, you can use a Gaussian function or a cubic spline function. You can also define a weight function with a user subroutine.

The Gaussian function is represented by:

$$\omega(r) = \frac{1}{(2\pi)^{3/2} r_c^3} exp\left(\frac{-r^2}{2r_c^2}\right)$$

The cubic spline function is represented by:

$$\omega(r) = \begin{cases} 4\left(\frac{r}{r_c} - 1\right)\left(\frac{r}{r_c}\right)^2 + \frac{2}{3}, & 0 < r < \frac{r_c}{2} \\ \frac{4}{3}\left(1 - -\frac{r}{r_c}\right)^3, & \frac{r_c}{2} \le r \le r_c \\ 0, & \text{otherwise} \end{cases}$$

Input File Usage: Use one of the following options:

**DAMAGE INITIATION*, WEIGHTING METHOD=UNIFORM (default)
**DAMAGE INITIATION*, WEIGHTING METHOD=GAUSS
**DAMAGE INITIATION*, WEIGHTING METHOD=CUBIC SPLINE
**DAMAGE INITIATION*, WEIGHTING METHOD=USER

Abaqus/CAE Usage: Specifying a weighting scheme for nonlocal averaging is not supported in Abaqus/CAE.

Smoothing the Normals of Individual Crack Facets Using Least-Squares Approximation

After the predicted crack propagation direction is obtained based on the nonlocal stress/strain averaging, a moving least-squares approximation by polynomials is used by default to further smooth out the crack normals. The least-squares approximation is applied to the normals of the individual facets in elements along the crack front

that satisfy the damage initiation criterion, as highlighted in *Figure 4*. This approximation provides a smoother crack surface (as shown in *Figure 5*), leading to a more accurate crack propagation direction.

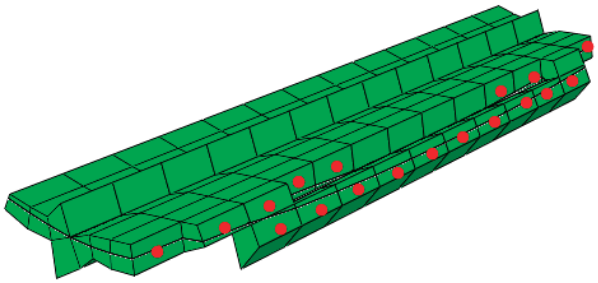


Figure 4: Cracked elements involved in the crack normal smoothing along the crack front.

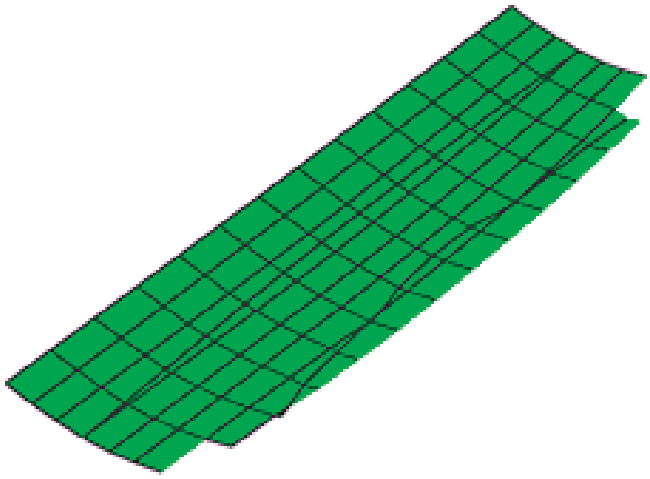


Figure 5: Smoothed crack surface.

You can use linear, quadratic, or cubic polynomial approximation for the moving least-squares approximation to smooth out the crack normals. You specify the number of terms in the polynomial. You can also suppress the least-squares approximation. In this case, the predicted crack propagation direction is determined based only on the nonlocal stress/strain averaging.

Input File Usage: Use one of the following options:

**DAMAGE INITIATION*, NPOLY=4
**DAMAGE INITIATION*, NPOLY=7 (default)
**DAMAGE INITIATION*, NPOLY=10
**DAMAGE INITIATION*, NPOLY=0 (suppress)

Abaqus/CAE Usage: Using the least-squares approximation to smooth out the crack normals is not supported in Abaqus/CAE.

Limiting the Elements Involved in Crack Normal Smoothing

At the beginning of the analysis, you can choose to include or exclude the preexisting crack facets in elements from the moving least-squares approximation to obtain the crack propagation direction. During the analysis, you can also limit the elements involved in the least-squares approximation. You can set the maximum allowed difference (in degrees) below which the normals of the crack facets are included in the moving least-squares approximation. The default is 70°.

Input File Usage: Use the following option to include the normals of preexisting crack facets:

**DAMAGE INITIATION*, INISMOOTH=YES (default)

Use the following option to exclude the normals of preexisting crack facets:

**DAMAGE INITIATION*, INISMOOTH=NO

Use the following option to limit the elements involved by setting a maximum allowed difference (in degrees):

**DAMAGE INITIATION*, ANGSMOOTH=*angle*

Abaqus/CAE Usage: Limiting the elements involved in crack normal smoothing is not supported in Abaqus/CAE.

Damage Evolution

The damage evolution law describes the rate at which the cohesive stiffness is degraded once the corresponding initiation criterion is reached. The general framework for describing the evolution of damage is conceptually similar to that used for damage evolution in surface-based cohesive behavior (*Contact Cohesive Behavior*).

A scalar damage variable, D, represents the averaged overall damage at the intersection between the crack surfaces and the edges of cracked elements. It initially has a value of 0. If damage evolution is modeled, D monotonically evolves from 0 to 1 upon further loading after the initiation of damage. The normal and shear stress components are affected by the damage according to

$$t_n = \begin{cases} (1-D)\,T_n, & T_n \geq 0 \\ T_n, & \text{otherwise (no damage to compressive stiffness);} \end{cases}$$

$$t_s = (1-D)\,T_s,$$

$$t_t = (1-D)\,T_t,$$

where T_n, T_s, and T_t are the normal and shear stress components predicted by the elastic traction-separation behavior for the current separations without damage.

To describe the evolution of damage under a combination of normal and shear separations across the interface, an effective separation is defined as

$$\delta_m = \sqrt{\langle \delta_n \rangle^2 + \delta_s^2 + \delta_t^2}.$$

Input File Usage: Use the following option to specify a damage evolution law:

**DAMAGE EVOLUTION*

The combination option **DAMAGE EVOLUTION*, TYPE=ENERGY, SOFTENING=EXPONENTIAL is not recommended to be used with XFEM.

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Traction Separation Laws**: **Maxpe Damage** or **Maxps Damage**: **Suboptions->Damage Evolution**

Use with User-Defined Damage Initiation Criterion

A separate damage evolution law should be specified for each damage initiation criterion defined in user subroutine *UDMGINI*. Each combination of a damage initiation criterion and a corresponding damage evolution law is referred to as a failure mechanism. Damage will accumulate for only one failure mechanism per element, corresponding to the mechanism whose damage initiation criterion was achieved first.

Input File Usage: Use the following options to specify damage evolution laws for multiple user-defined damage initiation criteria:

**DAMAGE INITIATION*, CRITERION=USER, FAILURE MECHANISMS=n
**DAMAGE EVOLUTION*, FAILURE INDEX=1
**DAMAGE EVOLUTION*, FAILURE INDEX=2
...
**DAMAGE EVOLUTION*, FAILURE INDEX=n

Abaqus/CAE Usage: Defining a user-defined damage initiation criterion is not supported in Abaqus/CAE.

Use with LaRC05 Criterion

You can specify four separate damage evolution laws, one for each of the four initiation mechanisms. Alternatively, you can specify fewer than four damage evolution laws. In this case, the initiation mechanisms that do not have a corresponding evolution law use the specified damage evolution law with the smallest failure index. Damage accumulates for only one failure mechanism per element, corresponding to the mechanism whose damage initiation criterion was achieved first.

Input File Usage: Use the following options to specify damage evolution laws for the LaRC05 damage initiation criterion:

**DAMAGE INITIATION*, CRITERION=LARC05
**DAMAGE EVOLUTION*, FAILURE INDEX=1
**DAMAGE EVOLUTION*, FAILURE INDEX=2
**DAMAGE EVOLUTION*, FAILURE INDEX=3
**DAMAGE EVOLUTION*, FAILURE INDEX=4

FAILURE INDEX=1, 2, 3, and 4 correspond to matrix cracking, fiber kinking, fiber splitting, and fiber tension, respectively.

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Fiber-Reinforced Composites->LaRC05 DamageSuboptions->Damage Evolution List**

Viscous Regularization in Abaqus/Standard

Models exhibiting various forms of softening behavior and stiffness degradation often lead to severe convergence difficulties in Abaqus/Standard. Viscous regularization of the constitutive equations defining cohesive behavior in an enriched element can be used to overcome some of these convergence difficulties. Viscous regularization damping causes the tangent stiffness matrix to be positive definite for sufficiently small time increments.

The approximate amount of energy associated with viscous regularization over the whole model is available using output variable ALLVD.

Input File Usage: Use the following option to specify viscous regularization:

**DAMAGE STABILIZATION*

Abaqus/CAE Usage: Property module: material editor: **Mechanical->Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Suboptions->Damage Stabilization Cohesive**

Output

Whole element variables:

STATUSXFEM Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked and 0.0 if the element contains no crack. If the element is partially cracked, the value of STATUSXFEM lies between 1.0 and 0.0.)

Surface variables (available only for propagating cracks modeled with first-order solid continuum elements):

CRKDISP Crack opening and relative tangential motions on cracked surfaces in enriched elements.

CSDMG Damage variable on cracked surfaces in enriched elements.

CRKSTRESS Remaining residual pressure and tangential shear stresses on cracked surfaces in enriched elements.

Applying the VCCT Technique to the XFEM-Based LEFM Approach

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- can be based on the principles of linear elastic fracture mechanics (LEFM);
- is more appropriate for brittle crack propagation;
- can also be used for simulating fatigue crack growth; and
- can be simultaneously used with the surface-based cohesive behavior approach (see *Contact Cohesive Behavior*) or the Virtual Crack Closure Technique (see *Crack Propagation Analysis*), which are best suited for modeling interfacial delamination.

By definition, the XFEM-based linear elastic fracture mechanics LEFM approach inherently requires the presence of a crack in the model since it is based upon the principles of linear elastic fracture mechanics—namely, an existing crack propagates when the energy released due to crack extension is equal to or more than the fracture toughness of the material (energy required to create new crack surfaces). The crack can be preexisting, or the crack can nucleate during the analysis. If there is no preexisting crack for a given enriched region, the XFEM-based LEFM approach is not activated until a crack nucleates.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- **SURFACE INTERACTION*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Crack Nucleation

If there is no pre-existing crack in the model, a crack can be nucleated as part of the analysis. The crack nucleation is governed by any of the available crack initiation criteria discussed in *Crack Initiation and Direction of Crack Extension*. After a crack nucleates in an enriched region, the XFEM-based LEFM criterion governs the subsequent propagation of the crack.

Input File Usage: Use the following option to specify the crack nucleation criterion as part of the material definition when there is no preexisting crack in an enriched region:

**DAMAGE INITIATION*, TOLERANCE=f_{tol}

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: f_{tol}

Crack Propagation/Extension

The formulae and laws that govern the behavior of the XFEM-based linear elastic fracture mechanics approach for crack propagation analysis are very similar to those used for modeling delamination along a known and partially bonded surface (see *VCCT Criterion*), where the strain energy release rate at the crack tip is calculated based on the modified Virtual Crack Closure Technique (VCCT). An existing crack is assumed to propagate when the energy released due to crack extension is equal to or greater than the fracture toughness of the material (energy required to create new crack surfaces). However, unlike the method described in *VCCT Criterion*, the XFEM-based LEFM approach can be used to simulate crack propagation along an arbitrary, solution-dependent path in the bulk material with or without an initial crack. You complete the definition of the crack propagation capability by defining a fracture-based surface behavior and specifying the fracture criterion in enriched elements.

Specifying When a Preexisting Crack Will Extend

For the discussion to follow, it is useful to define a ratio, *f*, of the energy release rate at the tip of a pre-existing crack to the fracture energy of the material. In the simple case of crack propagation in one of the pure modes (I, II, or III), this ratio may be defined as:

$$f = \frac{G_j}{G_{jC}},$$

where $j \in \{I, II, III\}$ is an index representing one of the pure modes. In the more general case of mixed-mode crack propagation, G_j and G_{jC} are replaced with G_{equiv} and G_{equivC}, respectively, which are measures of the energy release rate and fracture energy that are more appropriate under mixed-mode conditions and are defined more precisely later in the section. The ratio, *f*, provides a quantitative measure of the fracture criterion.

If there is a preexisting crack in an enriched region, the crack extends after an equilibrium increment when the fracture criterion, *f*, reaches the value 1.0 within a given tolerance:

$$1.0 \le f \le 1.0 + f_{tol}.$$

You can specify the tolerance f_{tol}. If $f > 1 + f_{tol}$, the time increment is cut back such that the crack extension criterion is satisfied. The default value of f_{tol} is 0.2.

Input File Usage: Use both of the following options:

**SURFACE BEHAVIOR*
**FRACTURE CRITERION*, TOLERANCE=f_{tol}, TYPE=VCCT

Abaqus/CAE Usage: Interaction module: **Interaction->Property->Create**, **Contact**, **Mechanical->Fracture Criterion**, **Tolerance**: f_{tol}

Fracture of Multiple Elements in an Unstable Crack Growth Analysis

For an unstable crack growth problem, sometimes it is more efficient to allow multiple elements at and ahead of a crack tip to fracture without excessively cutting back the increment size when the fracture criterion is satisfied. Abaqus/Standard activates this capability automatically if you specify an unstable growth tolerance, f^u_{tol}. In this case if the fracture criterion, *f*, is within the given unstable growth tolerance:

$$1.0 + f_{tol} \le f \le 1.0 + f^u_{tol},$$

where f_{tol} is the tolerance described earlier in this section, Abaqus/Standard immediately reduces the time increment size by default to a very small value, $\beta \triangle t_{\min}$. Reducing the time increment size allows more elements

to fracture until $f < 1$ for all the elements ahead of the crack tip. After this limit is reached, the time increment size is recovered automatically to a larger value, $\alpha \Delta t_{pre}$, where $\Delta t_{\min}$ is the minimum time increment allowed; Δt_{pre} is the time increment size prior to the unstable crack growth; and α, and β are scaling parameters. The default values of α and β are 0.5 and 2.0, respectively.

Use of a large unstable growth tolerance may sometimes lead to an overshoot of the peak load in the load displacement response of a structure. To prevent this, you may optionally specify the maximum number of cutbacks allowed, n, to be controlled by the regular tolerance, f_{tol}, prior to the activation of the unstable growth tolerance in an increment. The regular cutbacks ensure that the numerical solution does not over-predict the peak load that a structure containing a crack can sustain before the crack starts propagating. The default value of n is 0.

If you do not specify a value for the unstable growth tolerance, the default value is infinity. In this case the fracture criterion, *f*, for unstable crack growth is not limited by any upper bound value in the above equation.

Input File Usage: Use both of the following options:

**FRACTURE CRITERION*, UNSTABLE GROWTH TOLERANCE=f_{tol}^{u}
**CONTROLS*, TYPE=NO CUTBACK SCALING

Abaqus/CAE Usage: Use the following option to specify an unstable crack growth tolerance:

Interaction module: **Interaction**->**Property**->**Create**:
Contact:**Mechanical**->**Fracture Criterion**, Toggle on **Specify tolerance for unstable crack propagation**, select **Specify value**

Specifying the Crack Propagation Direction

You must specify the crack propagation direction when the fracture criterion is satisfied. The crack can extend at a direction normal to the direction of the maximum tangential stress, orthogonal to the element local 1-direction (see *Conventions*), or orthogonal to the element local 2-direction. By default, the crack propagates normal to the direction of the maximum tangential stress.

Input File Usage: Use one of the following options to specify the crack direction when the fracture criterion is satisfied:

**FRACTURE CRITERION*, NORMAL DIRECTION=MTS (default)

**FRACTURE CRITERION*, NORMAL DIRECTION=1

**FRACTURE CRITERION*, NORMAL DIRECTION=2

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical**->**Fracture Criterion**: **Direction of crack growth relative to local 1-direction**: **Maximum tangential stress**, **Normal**, or **Parallel**

Limiting the Crack Propagation Direction

If the crack direction normal to the maximum tangential stress is specified, you can limit the new crack propagation direction to within a certain angle (in degrees) of the previous crack propagation direction. The default is 85°.

Input File Usage: Use the following option to set the maximum allowed change in the crack propagation angle (in degrees):

**FRACTURE CRITERION*, ANGLEMAX=*max_angle*

Abaqus/CAE Usage: Limiting the crack propagation direction is not supported in Abaqus/CAE.

Nonlocal Smoothing of the Crack Normals

After the normals of the individual crack facets are obtained based on the fracture criterion defined above, you can use a moving least-squares approximation by polynomials to further smooth out the crack normals. The least-squares approximation is applied to the normals of the individual facets in elements along the crack front that satisfy the fracture criterion to obtain a more accurate crack propagation direction.

Input File Usage: **FRACTURE CRITERION*, POSITION=NONLOCAL

Abaqus/CAE Usage: Nonlocal smoothing of the crack normals is not supported in Abaqus/CAE.

Specifying the Approximation Used in the Least-Squares Approximation

You can use linear, quadratic, or cubic polynomial approximation for the moving least-squares approximation to smooth out the crack normals. You specify the number of terms in the polynomial.

Input File Usage: Use one of the following options:

**FRACTURE CRITERION*, NPOLY=4
**FRACTURE CRITERION*, NPOLY=7 (default)
**FRACTURE CRITERION*, NPOLY=10

Abaqus/CAE Usage: Nonlocal smoothing of the crack normals is not supported in Abaqus/CAE.

Specifying the Region of the Model Used for Nonlocal Smoothing of the Crack Normals

To control the range of elements used for nonlocal smoothing of the crack normals in the crack direction calculations, you can specify a radius, r_c, within which the elements around the crack tip along the crack front are included. The default radius is three times the typical element characteristic length along the crack front in the enriched region.

Input File Usage: **FRACTURE CRITERION*, R CRACK DIRECTION=r_c

Abaqus/CAE Usage: Nonlocal smoothing of the crack normals is not supported in Abaqus/CAE.

Mixed-Mode Behavior

Abaqus provides three common mode-mix formulae for computing the equivalent fracture energy release rate G_{equivC}: the BK law, the power law, and the Reeder law models. The choice of model is not always clear in any given analysis; an appropriate model is best selected empirically.

BK Law

The BK law model is described in Benzeggagh and Kenane (1996) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta},$$
$$G_{equiv} = G_I + G_{II} + G_{III}.$$

If there is a pre-existing crack in an enriched region, the crack extends after an equilibrium increment when the fracture criterion, $f = \frac{G_{equiv}}{G_{equivC}}$, reaches the value of 1.0 within a specified tolerance (as discussed earlier).

To define this model, you must provide G_{IC}, G_{IIC}, and η. This model provides a power law relationship combining energy release rates in Mode I, Mode II, and Mode III into a single scalar fracture criterion.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=BK

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical**->**Fracture Criterion**: **Mixed mode behavior**: **BK**, and enter the critical energy release rates in the data table

Power Law

The power law model is described in Wu and Reuter (1965) by the following formula:

$$\frac{G_{equiv}}{G_{equivC}} = \left(\frac{G_I}{G_{IC}} \right)^{a_m} + \left(\frac{G_{II}}{G_{IIC}} \right)^{a_n} + \left(\frac{G_{III}}{G_{IIIC}} \right)^{a_o}.$$
$$G_{equiv} = G_I + G_{II} + G_{III}.$$

If there is a pre-existing crack in an enriched region, the crack extends after an equilibrium increment when the fracture criterion, $f = \frac{G_{equiv}}{G_{equivC}}$, reaches the value of 1.0 within a specified tolerance (as discussed earlier).

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}, a_m, a_n$, and a_o.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=POWER

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical**->**Fracture Criterion**: **Mixed mode behavior**: **Power**, and enter the critical energy release rates in the data table

Reeder Law

The Reeder law model is described in Reeder et al. (2002) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta} +$$
$$(G_{IIIC} - G_{IIC}) \left(\frac{G_{III}}{G_{II}+G_{III}} \right) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta},$$
$$G_{equiv} = G_I + G_{II} + G_{III}.$$

If there is a pre-existing crack in an enriched region, the crack extends after an equilibrium increment when the fracture criterion, $f = \frac{G_{equiv}}{G_{equivC}}$, reaches the value of 1.0 within a specified tolerance (as discussed earlier).

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC},$ and η. The Reeder law is best applied when $G_{IIC} \neq G_{IIIC}$; when $G_{IIC} = G_{IIIC}$, the Reeder law reduces to the BK law. The Reeder law applies only to three-dimensional problems.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=REEDER

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical**->**Fracture Criterion**: **Mixed mode behavior**: **Reeder**, and enter the critical energy release rates in the data table

Defining Variable Critical Energy Release Rates

You can define a VCCT criterion with varying energy release rates by specifying the critical energy release rates at the nodes.

If you indicate that the nodal critical energy rates will be specified, any constant critical energy release rates you specify are ignored and the critical energy release rates are interpolated from the nodes. The critical energy release rates must be defined at all nodes in the enriched region.

Input File Usage: Use both of the following options:

**FRACTURE CRITERION*, TYPE=VCCT, NODAL ENERGY RATE
**NODAL ENERGY RATE*

Abaqus/CAE Usage: Defining a VCCT criterion with varying energy release rates is not supported in Abaqus/CAE.

Enhanced VCCT Criterion

The formulae and laws governing the behavior of the enhanced VCCT criterion are very similar to those used for the VCCT criterion. However, unlike the VCCT criterion, the onset and growth of a crack can be controlled by two different critical fracture energy release rates: G_C and G_C^P. In a general case involving Mode I, II, and III fracture, when the fracture criterion is satisfied; i.e,

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0,$$

the traction on the two surfaces of the cracked element is ramped down over the separation with the dissipated strain energy equal to the critical equivalent strain energy required to propagate the crack, G^P_{equivC}, rather than the critical equivalent strain energy required to initiate the separation, G_{equivC}. The formulae for calculating G^P_{equivC} are identical to those used for G_{equivC} for different mixed-mode fracture criteria.

Input File Usage: Use both of the following options:

**SURFACE BEHAVIOR*
**FRACTURE CRITERION*, TYPE=ENHANCED VCCT

Abaqus/CAE Usage: Specifying the enhanced VCCT criterion is not supported in Abaqus/CAE.

Fatigue Crack Growth Criterion Based on the Principles of LEFM

If you specify the fatigue crack growth criterion, progressive crack growth at the enriched elements subjected to sub-critical cyclic loading can be simulated. This criterion can be used only in the general fatigue crack growth approach (*Linear Elastic Fatigue Crack Growth Analysis*). A fatigue crack growth analysis step can be the only step, can follow a general static step, or can be followed by a general static step. You can include multiple fatigue crack growth analysis steps in a single analysis. If you perform a fatigue analysis in a model without a preexisting crack, you must precede the fatigue step with a static step that nucleates a crack, as discussed in *Crack Propagation/Extension*.

The onset and fatigue crack growth are characterized by using the Paris law, which relates the fracture energy release rate or the stress intensity factor to crack growth rates. Different forms of Paris law are available in Abaqus. For details, see *Fatigue Crack Growth Using the Paris Law*

Input File Usage: Use both of the following options:

**SURFACE BEHAVIOR*
**FRACTURE CRITERION*, TYPE=FATIGUE

Abaqus/CAE Usage: Specifying a fatigue crack growth criterion is not supported in Abaqus/CAE.

Linear Elastic Fatigue Crack Growth Analysis

The fracture energy release rates or the stress intensity factors at the crack tips in the enriched elements are calculated based on the above mentioned VCCT technique. The process of fatigue crack growth is based on a damage extrapolation technique that relies on fracturing at least one enriched element ahead of the current crack tip (once the onset criterion is satisfied) with a corresponding increase in the number of cycles. Specifically, if the criterion for the onset of the crack growth is satisfied at any crack tip in the enriched element at the end of a completed cycle, N, Abaqus/Standard extends the crack length, a_N, by fracturing at least one enriched element ahead of the crack tip/front. This increase in the crack length corresponds to an increase in the cycle count from N to $N + \Delta N$, where ΔN is computed based on the Paris law. For additional information on how a crack extends over an incremental number of cycles, see *Damage Extrapolation Technique*.

If $G_{max} > G_{pl}$, the enriched elements ahead of the crack tips will be fractured by increasing the cycle number count, ΔN, by one only.

For information on how to accelerate the fatigue crack growth analysis and to provide a smooth solution for the crack front, see *Controlling Element Fracture*.

Viscous Regularization for the XFEM-Based LEFM Approach

The simulation of structures with unstable propagating cracks is challenging and difficult. Nonconvergent behavior may occur from time to time. Localized damping is included for the XFEM-based LEFM approach by using the viscous regularization technique. Viscous regularization damping causes the tangent stiffness matrix of the softening material to be positive for sufficiently small time increments.

Input File Usage: Use one of the following options

**FRACTURE CRITERION*, TYPE=VCCT, VISCOSITY=μ

**FRACTURE CRITERION*, TYPE=ENHANCED VCCT, VISCOSITY=μ

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical**->**Fracture Criterion**: **Viscosity**: μ

Output

Whole element variables:

ENRRTXFEM	All components of strain energy release rate when linear elastic fracture mechanics with the extended finite element method is used.
CYCLEXFEM	Number of cycles to fracture at the enriched element.
CYCLEINIXFEM	Minimum number of cycles needed to satisfy the condition for the onset of fatigue crack growth at an enriched element.

Surface variables (available only for propagating cracks modeled with first-order solid continuum elements):

CRKDISP	Crack opening and relative tangential motions on cracked surfaces in enriched elements.
CSDMG	Damage variable on cracked surfaces in enriched elements.
CRKSTRESS	Remaining residual pressure and tangential shear stresses on cracked surfaces in enriched elements.

References

- Benzeggagh, M., and M. Kenane, "Measurement of Mixed-Mode Delamination Fracture Toughness of Unidirectional Glass/Epoxy Composites with Mixed-Mode Bending Apparatus," *Composite Science and Technology*, vol. 56 439, 1996.
- Irwin, G. R., "Linear Fracture Mechanics Fracture Transition, and Fracture Control," *Engineering Fracture Mechanics*, vol. 1, pp. 241–257, 1968.
- Reeder, J., S. Kyongchan, P. B. Chunchu, and D. R.. Ambur, "Postbuckling and Growth of Delaminations in Composite Plates Subjected to Axial Compression" 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, vol. 1746, p. 10, 2002.
- Wu, E. M., , and R. C. Reuter Jr., "Crack Extension in Fiberglass Reinforced Plastics," T and M Report, University of Illinois, vol. 275, 1965.

Contact Interaction of Cracked Element Surfaces

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- allows contact interaction of cracked element surfaces, including thermal interaction and pore fluid interaction, based on a small-sliding formulation or on a finite-sliding formulation within the general contact framework.

Contact interaction of cracked element surfaces, including thermal interaction and pore fluid interaction, is allowed based on a small-sliding formulation or on a finite-sliding formulation within the general contact framework.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- **SURFACE INTERACTION*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Defining Contact of Cracked Element Surfaces Using a Small-Sliding Formulation

When an element is cut by a crack, the compressive behavior of the crack surfaces has to be considered. The formulation that governs the compressive behavior is very similar to that used for surface-based small-sliding penalty contact (*About Mechanical Contact Properties*). A more general small-sliding surface interaction model between the cracked element surfaces can be defined in user subroutine *UXFEMCRACK*.

For an element intersected by a stationary crack or a moving crack with the linear elastic fracture mechanics approach, it is assumed that the elastic cohesive strength of the cracked element is zero. Therefore, compressive behavior of the crack surfaces is fully defined with the above options when the crack surfaces come into contact. For a moving crack with the cohesive segments method, the situation is more complex; traction-separation cohesive behavior as well as compressive behavior of the crack surfaces are involved in a cracked element. In the contact normal direction, the pressure-overclosure relationship governing the compressive behavior between the surfaces does not interact with the cohesive behavior, since they each describe the interaction between the surfaces in a different contact regime. The pressure-overclosure relationship governs the behavior only when the crack is “closed”; the cohesive behavior contributes to the contact normal stress only when the crack is “open” (that is, not in contact).

If the elastic cohesive stiffness of an element is undamaged in the shear direction, it is assumed that the cohesive behavior is active. Any tangential slip is assumed to be purely elastic in nature and is resisted by the elastic cohesive strength of the element, resulting in shear forces. If damage has been defined, the cohesive contribution to the shear stresses starts degrading with damage evolution. Once maximum degradation has been reached, the cohesive contribution to the shear stresses is zero. The friction model activates and begins contributing to the shear stresses.

Input File Usage: Use the following options to define contact of crack surfaces using a small-sliding formulation. The associated surface interaction property defines the cohesive and compressive regimes for the opposing crack surfaces.

**ENRICHMENT*, INTERACTION=*interaction_property_name*
**SURFACE INTERACTION*, NAME=*interaction_property_name*
**SURFACE BEHAVIOR*

Use the following options to define a more general small-sliding surface interaction of cracked element surfaces using user subroutine *UXFEMCRACK*:

**ENRICHMENT*, INTERACTION=*interaction_property_name*
**SURFACE INTERACTION*, NAME=*interaction_property_name*, *UXFEMCRACK*

Abaqus/CAE Usage: Interaction module: crack editor: toggle on **Specify contact property**

Defining a user-defined small-sliding surface interaction using subroutine *UXFEMCRACK* is not supported in Abaqus/CAE.

Defining Contact of Cracked Element Surfaces Using General Contact

A general-purpose finite-sliding contact capability is available for XFEM-based crack surfaces within the general contact framework in Abaqus/Standard (*About General Contact in Abaqus/Standard*). Contact pairs do not support XFEM-based crack surfaces. Crack surfaces can be included in the general contact domain to participate in contact within the compressive regime. Contact between crack surfaces and contact between crack surfaces and other types of surfaces in the model can be considered. The general contact algorithm includes crack surfaces when the default all-inclusive exterior surface defines the contact domain. Alternatively, you can specify pairwise inclusions and exclusions to control more precisely the regions of the model that can potentially come into contact. In both cases the enrichment regions where cracks occur and participate in general contact have to be associated with at least one XFEM-based crack surface. In the former case the association of an enrichment region with an XFEM-based crack surface triggers general contact on cracks in that enrichment region. In the latter case the named XFEM-based crack surface can be explicitly used with contact inclusions and exclusions akin to other types of surfaces to precisely define the general contact domain (*Defining the General Contact Domain*).

General contact always takes precedence over the small-sliding contact formulation when both are included to model contact between opposing surfaces of a crack within the same analysis. The small-sliding formulation cannot be used to model contact between the crack surfaces and other surfaces; in that case, only general contact supports contact interactions. General contact is applicable only for a crack propagation analysis.

Input File Usage: Use the following options to associate two enrichment regions with the same XFEM-based crack surface:

**ENRICHMENT*, NAME=*region-1*, TYPE=PROPAGATION CRACK, ELSET=*set-1*
**ENRICHMENT*, NAME=*region-2*, TYPE=PROPAGATION CRACK, ELSET=*set-2*
**SURFACE*, TYPE=XFEM, NAME=*crack_surf*
region-1
region-2

Use the following options to define an all-inclusive surface that includes XFEM-based crack surfaces to model contact interactions:

**CONTACT*
**CONTACT INCLUSIONS*, ALL EXTERIOR
**SURFACE INTERACTION*, NAME=*interaction_property_name*

The following example shows a named XFEM-based crack surface considered for contact with the default all-inclusive exterior surface:

**CONTACT*
**CONTACT INCLUSIONS*
crack_surf,
**SURFACE INTERACTION*, NAME=*interaction_property_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction: General contact (Standard): Included surface pairs: All* with self**

Thermal Contact Interaction at Cracked Element Surfaces

Optionally, you can specify thermal contact interaction properties (gap conductance, gap radiation, and gap heat generation) in a contact property definition for cracked element surfaces using the small-sliding formulation or using the general contact finite-sliding formulation. You can include all three types of thermal properties in the same contact property definition. General contact always takes precedence over the small-sliding contact formulation when both are included to model thermal contact between opposing surfaces of a crack within the same analysis.

Input File Usage: Use the following options to define thermal contact properties:

**SURFACE INTERACTION*, NAME=*interaction_property_name*
**GAP CONDUCTANCE*
**GAP RADIATION*
**GAP HEAT GENERATION*

Abaqus/CAE Usage: Interaction module: contact property editor: **Thermal**->**Thermal Conductance**, **Heat Generation**, and/or **Radiation**

Defining the Constitutive Response of Fluid Flow within the Cracked Element Surfaces

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- enables the modeling of discontinuities in the fluid pressure field as well as fluid flow within the cracked element surfaces;
- is typically used in geotechnical applications, where fluid flow continuity within the cracked element surfaces and across the cracked element surfaces must be maintained;
- enables fluid pressure on the cracked element surface to contribute to its mechanical behavior, which enables the modeling of hydraulically driven fracture;
- can include the heat transport due to thermal conductance and radiation across the cracked element surfaces as well as thermal convection within the cracked element surfaces; and
- allows multiple fluid definition.

The formulae and laws that govern the behavior of fluid flow within the XFEM-based cracked element surfaces are very similar to those used for fluid flow within the cohesive element gap (*Defining the Constitutive Response of Fluid within the Cohesive Element Gap*). The similarities extend to the traction-separation model, damage initiation criteria, damage evolution law, and the fluid flow behavior. The fluid constitutive response includes the tangential flow within the cracked element surfaces and the normal flow across the cracked element surfaces due to caking or fouling effects in the enriched elements.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- **FLUID LEAKOFF*
- **GAP FLOW*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Tangential Flow

The tangential flow within the cracked element surfaces can be modeled with either a Newtonian or power-law model. By default, there is no tangential flow of pore fluid within the cracked element surfaces. To allow tangential flow, define a gap flow property in conjunction with the pore fluid material definition.

In the case of a Newtonian fluid the volume flow rate density vector is given by the expression

$$\mathbf{q}d = -k_t \nabla p,$$

where k_t is the tangential permeability (the resistance to the fluid flow), ∇p is the pressure gradient along the cracked element surfaces, and d is the opening of the cracked element surfaces.

Abaqus defines the tangential permeability, or the resistance to flow, according to Reynold's equation:

$$k_t = \frac{d^3}{12\mu},$$

where μ is the fluid viscosity and d is the opening of the cracked element surfaces. You can also specify an upper limit on the value of k_t.

In the case of a power law fluid the constitutive relation is defined as

$$\tau = K\dot{\gamma}^{\alpha},$$

where τ is the shear stress, $\dot{\gamma}$ is the shear strain rate, K is the fluid consistency, and α is the power law coefficient. Abaqus defines the tangential volume flow rate density as

$$\mathbf{q}d = -\left(\frac{2\alpha}{1+2\alpha}\right)\left(\frac{1}{K}\right)^{\frac{1}{\alpha}}\left(\frac{d}{2}\right)^{\frac{1+2\alpha}{\alpha}}\|\nabla p\|^{\frac{1-\alpha}{\alpha}}\nabla p,$$

where d is the opening of the cracked element surfaces.

By default, the gap between the cracked element surfaces has an initial opening of 0.002 in both a Newtonian fluid and a power law fluid. However, you can specify this opening directly.

Input File Usage: Use the following option to define the tangential flow in a Newtonian fluid:

**GAP FLOW*, TYPE=NEWTONIAN, KMAX

Use the following option to define the tangential flow in a power law fluid:

**GAP FLOW*, TYPE=POWER LAW

Use the following option to define the initial gap opening directly:

**SECTION CONTROLS*, INITIAL GAP OPENING

Abaqus/CAE Usage: Use the following option to define the tangential flow in a Newtonian fluid:

Property module: material editor: **Other->Pore Fluid->Gap Flow**: Type: **Newtonian**: Toggle on **Maximum Permeability** and enter the value of $k_{\max}$

Use the following option to define the tangential flow in a power law fluid:

Property module: material editor: **Other->Pore Fluid->Gap Flow**: Type: **Power law**

An initial gap opening is not supported in Abaqus/CAE.

Normal Flow across the Cracked Element Surfaces

You can permit normal flow by defining a fluid leak-off coefficient for the pore fluid material. This coefficient defines a pressure-flow relationship between the phantom nodes located at the cracked element edges and cracked element surfaces. The fluid leak-off coefficients can be interpreted as the permeability of a finite layer of material on the cracked element surfaces, as shown in *Figure 1*.

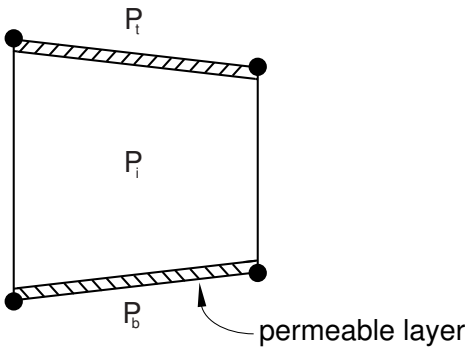


Figure 1: Leak-off coefficient interpretation as a permeable layer.

The normal flow is defined as

$$q_t = c_t \left(p_i - p_t\right)$$

and

$$q_b = c_b \left(p_i - p_b\right),$$

where q_t and q_b are the flow rates into the top and bottom surfaces of a cracked element, respectively; p_i is the pressure at the phantom node located at the cracked element edge; and p_t and p_b are the pore pressures on the top and bottom surfaces of a cracked element, respectively. You can optionally define leak-off coefficients as functions of temperature and field variables.

Alternatively, you can use user subroutine *UFLUIDLEAKOFF* to define more complex leak-off behavior, including the ability to define a time accumulated resistance, or fouling, through the use of solution-dependent state variables.

Input File Usage: Use the following option to define the leak-off coefficients:

**FLUID LEAKOFF*

Use the following option to define leak-off coefficients as functions of temperature and field variables:

**FLUID LEAKOFF*, DEPENDENCIES

Use the following option to define more complex leak-off behavior in user subroutine *UFLUIDLEAKOFF*:

**FLUID LEAKOFF*, USER

Abaqus/CAE Usage: Use the following option to define the leak-off coefficients:

Property module: material editor: **Other**->**Pore Fluid**->**Fluid Leakoff**: Type: **Coefficients**

Use the following option to define leak-off coefficients as functions of temperature and field variables:

Property module: material editor: **Other**->**Pore Fluid**->**Fluid Leakoff**: Type: **Coefficients**: Toggle on **Use temperature-dependent data** and select the number of field variables.

Use the following option to define more complex leak-off behavior in user subroutine *UFLUIDLEAKOFF*:

Property module: material editor: **Other->Pore Fluid->Fluid Leakoff**: Type: **User**

Thermal Effect of Fluid Flow within the Cracked Surfaces

If the thermal effect of the fluid flow within the cracked element surfaces is considered, the heat transport response comprises the tangential heat flux convection within the cracked surfaces, the normal heat convection between the gap fluid and the fracture surfaces, and the normal heat flux convected by fluid flow across the cracked element surfaces.

Optionally, you can include the normal heat convection between the gap fluid and the fracture surfaces.

The normal heat convection is defined as

$$q_t = h_t \left(\theta_i - \theta_t\right)$$

and

$$q_b = h_b \left(\theta_i - \theta_b\right),$$

where q_t and q_b are the heat fluxes into the top and bottom surfaces of a cracked element, respectively; θ_i is the temperature at the phantom node located on the cracked element edge; and θ_t and θ_b are the temperatures on the top and bottom surfaces of a cracked element, respectively.

Input File Usage: Use the following options to specify the heat capacity and conductivity properties of the fluid:

**SPECIFIC HEAT*, PORE FLUID

**CONDUCTIVITY*, PORE FLUID

Use the following option to define the gap film coefficients:

**GAP CONVECTION*

Abaqus/CAE Usage: Defining the heat capacity, conductivity properties of the fluid, and the gap film coefficients is not supported in Abaqus/CAE.

Defining Multiple Fluid Flows within the Cracked Surfaces

Abaqus allows the use of multiple fluids in an analysis, up to a maximum of four fluid types. This capability is useful in situations where the spatial distribution of the different fluid types is known (that is, precomputed) at all times during the analysis. In other words, the type of fluid at each node in the domain is known at all times during the analysis.

The definition of multiple fluids in an analysis involves associating each fluid with a unique integer and a specific predefined field variable. This integer acts as a fluid identifier. You can utilize this fluid identifier to specify the viscosity of each fluid. Only the fluid viscosity can be different for different fluids. All other properties (e.g., density, thermal) are assumed to be same for all the fluids. The field variable associated with a fluid type allows you to predefine the fluid type at a node as a function of time using amplitude definitions, or as functions of both time and space utilizing user-subroutines *UFIELD* or *USDFLD*.

You specify a value for each field variable, which is associated with a fluid identifier, at each node. It is recommended that you specify a value for the field variable (FV) such that $0 \leq FV \leq 1.0$Abaqus assigns the

fluid type at an integration point to be the one for which the field variable has the maximum value at that point. If all field variables have the same numerical value at an integration point, the first fluid (FLUID ID = 1) is assumed to be active.

Input File Usage: **GAP FLOW*, FLUID ID=*Fluid Id number*, DEFINITION=NEWTONIAN or POWER LAW or HERSCHEL-BULKLEY or BINGHAM PLASTIC
**FIELD*, VARIABLE=*Field variable number*
**FLUID ID FIELD*
Fluid Id number, *Field variable number*

Abaqus/CAE Usage: Specifying multifluid flow is not supported in Abaqus/CAE.

Output

The following whole element output variables are available only when fluid flow is enabled within the cracked enriched element surfaces:

GFVRXFEM Gap fluid volume rate of the enriched element.

CRDCUTXFEM Crack midpoint coordinates at the element edges of the enriched element.

PFOPENXFEM Fracture opening of the enriched element.

PFOPENXFEMCOMP Fracture opening at the element edges of the enriched element.

PORPRES Fluid pressure of the enriched element.

PORPRESCOMP Fluid pressure at the element edges of the enriched element.

LEAKVRTXFEM Leak-off flow rate at the top of the enriched element.

LEAKVRBXFEM Leak-off flow rate at the bottom of the enriched element.

ALEAKVRTXFEM Accumulated leak-off flow volume at the top of the enriched element.

ALEAKVRBXFEM Accumulated leak-off flow volume at the bottom of the enriched element.

The following surface output variables are available only when fluid flow is enabled within the cracked enriched element surfaces:

GFVR Fluid volume rate within the cracked surfaces in the enriched element.

PORPRES Pore pressure within the cracked surfaces in the enriched element.

PORPRESURF Pore pressure on the cracked surfaces in the enriched element.

LEAKVR Leak-off flow rate on the cracked surfaces in the enriched element.

ALEAKVR Accumulated leak-off flow volume on the cracked surfaces in the enriched element.

The following surface output variable is available only when both fluid flow and thermal convection are enabled within the cracked enriched element surfaces:

PORTEMP Temperature within the cracked surfaces in the enriched element.

Modeling Slurry Flow

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- can include slurry transport and placement within the cracked element surfaces;
- is typically used in geotechnical applications where the continuity of the slurry mixture and its components within the cracked element surfaces and across the cracked element surfaces must be maintained;
- enables modeling of an additional resistance layer on the cracked element surfaces to model fluid leakoff into the formation, with or without considering the evolution of filter cake;
- enables fluid pressure on the cracked element surfaces to contribute to its mechanical behavior, which enables the modeling of hydraulically driven fracture;
- enables proppant settling due to gravity and wall drag effects;
- enables the modeling of hydraulic fracture propped by solid particles; and
- allows multiple fluid definition.

The slurry transport and placement capabilities in Abaqus/Standard model transport of solids (proppants)-laden Newtonian and non-Newtonian slurries within the cracked surfaces. The slurry advects into the crack region, as it opens up due to hydraulic pressure. Subsequently, the particles in the slurry settle down within the crack through a process that is mediated by wall drag and gravity effects. The particles help prop open the crack if it tries to close down due to changes in external conditions.

This complex set of phenomena within the cracked surfaces can be described in terms of the mean tangential flow of the slurry mixture, the advection of the proppant particles with the carrying fluid, and the settling of the proppant particles within the crack due to wall drag and gravity effects. The process is governed by the continuity of the slurry mixture (a mixture of the proppant particles and the carrying fluid) that describes the mean flow of the slurry, the continuity of the carrier fluid, a description of the settling phenomenon in terms of the relative velocity of the particles with respect to the carrying fluid, and leakoff (normal flow) of the carrying fluid into the surrounding formation.

References:

- **ENRICHMENT*
- **ENRICHMENT ACTIVATION*
- **CONTACT*
- **FLUID LEAKOFF*
- **GAP FLOW*
- *Using the extended finite element method to model fracture mechanics*
- *Extended finite element method (XFEM)*

Tangential Slurry Flow

The continuity equation for the mean flow of the slurry mixture within the crack can be written as:

$$\frac{\partial d}{\partial t} + \nabla \cdot (d\boldsymbol{V}_m) + q_l = 0,$$

where d is the opening of the cracked element surfaces, $\boldsymbol{V}_m$ is the *mean velocity* of slurry, and q_l is the leak-off flow rate. The continuity equation for the carrier fluid can be written as

$$\frac{\partial[(1-C_v)d]}{\partial t} + \nabla\cdot[(1-C_v)d\boldsymbol{V}_w] + q_l = 0,$$

where C_v is the concentration of proppant particles in the slurry, defined as the volume ratio of the solid phase to that of the slurry phase (henceforth referred to as the slurry concentration), and $\boldsymbol{V}_w$ is the velocity vector of the carrier fluid. In formulating the above equation, it is assumed that the carrier fluid and solid particles move with different velocity vectors, $\boldsymbol{V}_w$ and $\boldsymbol{V}_s$, respectively, and that they are related to the mean slurry velocity and a *settling (or slip) velocity*, $\boldsymbol{V}_{s/w}$, as

$$\begin{cases} \boldsymbol{V}_w = \boldsymbol{V}_m + C_v\boldsymbol{V}_{s/w} \\ \boldsymbol{V}_s = \boldsymbol{V}_m - (1-C_v)\,\boldsymbol{V}_{s/w} \end{cases},$$

where the settling velocity $\boldsymbol{V}_{s/w} = \boldsymbol{V}_w - \boldsymbol{V}_s$ represents the relative velocity of the proppant particles with respect to the carrier fluid. Using the above velocity split, the carrier fluid continuity equation can be rewritten as

$$\frac{\partial[(1-C_v)d]}{\partial t} + \nabla\cdot[(1-C_v)d\boldsymbol{V}_m] + q_l + \nabla\cdot\left[C_v(1-C_v)d\boldsymbol{V}_{s/w}\right] = 0.$$

The above system of equations is augmented by a relation of the following form, which relates the mean slurry flux, $\mathbf{q}$, and the gradient of the fluid pressure field (assuming non-Newtonian constitutive behavior of the slurry),

$$\mathbf{q}d = f\left(K, \alpha, \tau_0, d, \nabla p\right),$$

where K is the fluid consistency, α is the power law coefficient, τ_0 is the yield stress, and d is the opening of the cracked element surfaces. The above system of equations helps determine the fluid pressure field that drives the slurry flow through the gap, the mean slurry flow velocity field, and the slurry concentration, C_v (nodal degree of freedom 31), accounting for both leakoff and settling.

Abaqus assumes that the homogenized slurry flow properties can be defined in a manner that is similar to the corresponding definitions for the flow of a pure fluid within the cracked surfaces (*Defining the Constitutive Response of Fluid Flow within the Cracked Element Surfaces*), except that the flow properties can be functions of the slurry concentration.

Input File Usage: Use the following option to define the tangential flow of a slurry mixture with a specified fluid type:

**GAP FLOW*, TYPE=*type_name*, SLURRY

Multifluid Slurry Flow

Abaqus allows the use of multiple carrier fluids in an analysis that models slurry transport and placement. As described earlier (*Defining Multiple Fluid Flows within the Cracked Surfaces*), only the fluid viscosity can be different for different carrier fluids. All other properties (for example, density, thermal) are assumed to be same for all the carrier fluids.

Proppant Settling

By default, Abaqus assumes that the particle settling phenomena is mainly due to gravity (solid particles and carrier fluid usually have different densities) and wall-drag effects (hindrance due to the narrowness of the gap). However you might suppress the settling completely (thereby modeling proppant transport due to advection only), or you can define the settling velocity field directly through user subroutine *USETTLING*.

Input File Usage: Use one of the following option to account for settling effects using the Abaqus built-in model:

**GAP FLOW*, TYPE=*type_name*, SLURRY

**GAP FLOW*, TYPE=*type_name*, SLURRY, SETTLING

Use the following option to suppress settling effects:

**GAP FLOW*, TYPE=*type_name*, SLURRY, SETTLING=NONE

Use the following option to define the settling velocity field directly, through user subroutine *USETTLING*:

**GAP FLOW*, TYPE=*type_name*, SLURRY, SETTLING=USER

Abaqus/CAE Usage: Specifying settling is not supported in Abaqus/CAE.

Crack Opening Particle Size Ratio

By default, Abaqus assumes that the ratio of the opening between the cracked element surfaces to the particle size must be greater than 1.0 for the slurry particles to enter the crack. Slurry calculations inside the cracked surfaces are not activated by default if the calculated ratio between these quantities is less than 1.0. Alternatively, you can specify a nondefault minimum ratio of the crack opening to the particle size.

Input File Usage: Use the following option to define the density of the carrier fluid, the density of the proppant particles, and the diameter of the proppant particles (it is assumed that all the proppant particles in the slurry have the same diameter), respectively:

**DENSITY*, PORE FLUID, SLURRY

Use the following option to define the ratio of the minimum gap opening to the particle size directly:

**SECTION CONTROLS*, MIN GAP PARTICLE RATIO

Abaqus/CAE Usage: Defining the density of the carrier fluid and the proppant particles, and the gap-particle diameter ratio of the slurry is not supported in Abaqus/CAE.

Stability of Tangential Flow

A Petrov-Galerkin scheme (see *Convection/diffusion*) is used to discretize the governing equation for the slurry concentration because the slurry transport is a phenomenon dominated by advection. Although diffusion effects are assumed to be negligible, you can define a small amount of diffusivity to provide numerical stabilization.

Input File Usage: Use the following options to define the diffusivity of proppant particles in a slurry flow:

**DIFFUSIVITY*, SLURRY

Abaqus/CAE Usage: Defining the diffusivity of proppant particles of a slurry flow is not supported in Abaqus/CAE.

A small amount of diffusivity is generally recommended for a slurry transport analysis. You must be careful to select a value of the diffusivity coefficient such that the fundamental nature of the transport problem is not modified from convection-dominated to diffusion-dominated. A good measure of the relative dominance of convective versus diffusive contributions is the local Péclet number, γ, which is defined as:

$$\gamma = |\mathbf{v}| \frac{\Delta l}{D}$$

where $|\mathbf{v}|$ is the magnitude of the velocity vector, Δl is a characteristic element length in the direction of flow, and D is the diffusivity of the proppants. Large values of γ indicate that convection dominates over diffusion on the spatial scale defined by the element size Δl.

Normal Flow with Slurry Transport

The normal flow across the cracked element surfaces can be modeled using one of the following approaches:

- A simple leak-off behavior that require leak-off coefficients on the top and bottom cracked surfaces to define the flux of the normal flow, or
- A physics based filtration model that accounts for the formation and evolution of filter cake layers on the two cracked element surfaces.

The physics-based approach has following flavors (*Outmans, 1963*), which are applicable only when the slurry concentration degree of freedom is active in the analysis:

- A dynamic filtration model that assumes the filter cake to be compressible.
- A static filtration model that assumes the filter cake to be incompressible.

Therefore you can model the normal flow across the cracked element surfaces using one of the following approaches:

- the simple leak-off model in which you define the leak-off coefficients directly;
- a physics-based model that accounts for the evolution of the filter cake thickness, filter cake viscosity, and filter case permeability; or
- through user subroutine *UFLUIDLEAKOFF*, which allows you to define more complex leak-off behavior (including the ability to define a time-accumulated resistance, or fouling, through the use of solution-dependent state variables).

Simple Leak-Off Behavior

The simple leak-off behavior assumes that the flux of the normal flow across the top and the bottom surfaces is defined by a simple pressure-flow relationship (for more information, see *Normal Flow with Slurry Transport*).

The simple leak-off model does not account for the dependence of the leak-off behavior on the slurry concentration but might still provide a reasonable description of the normal flow across the cracked element surfaces in some situations.

Input File Usage: Use the following option to define the fluid leak-off coefficients at the top and the bottom surfaces directly, (optionally) as tabular functions of temperature:

**FLUID LEAKOFF*

Use the following option to define the fluid leak-off coefficients at the top and the bottom surfaces directly, (optionally) as tabular functions of temperature and field variables:

**FLUID LEAKOFF*, DEPENDENCIES

Use the following option to define the fluid leak off coefficients using user subroutine *UFLUIDLEAKOFF*:

**FLUID LEAKOFF*, USER

Abaqus/CAE Usage: Property module: material editor: **Other->Pore Fluid->Fluid Leakoff**: Type: **Coefficients**

Property module: material editor: **Other->Pore Fluid->Fluid Leakoff**: Type: **Coefficients**: Toggle on **Use temperature-dependent data** and select the number of field variables.

Property module: material editor: **Other->Pore Fluid->Fluid Leakoff**: Type: **User**

Physics-Based Leak-Off Behavior

The physics-based leak-off behavior is based upon *Outmans (1963)* filtration models, which assume the following:

- the filter cake is a porous medium;
- the fluid flow through the filter cake is governed by Darcy's law; and
- the rate of increase of the thickness of the filter cake layer is proportional to the flux of fluid through the filter cake.

Abaqus assumes the filter cake to be compressible in a dynamic filtration model, while it is assumed to be incompressible in a static filtration model.

In the discussion to follow, p_f and p_p represent the (slurry) fluid pressure and formation pore fluid pressure, respectively.

For the dynamic filtration model, the porosity of the filter cake is defined as:

$$\varepsilon_h = \varepsilon_0 \exp\left[-\alpha\left(p_f - p_p\right)\right]$$

where $\varepsilon_0 = C_f\varepsilon^*$, $\varepsilon^* = 0.68$ is the porosity corresponding to packing of mono-dispersed spherical particles, C_f represents a correction factor that accounts for particle size variations, and α is a compressibility factor. The case $\alpha = 0$ represents incompressible static filter cake behavior, which implies that $\varepsilon_h = \varepsilon_0$. You specify both C_f and α to define a dynamic filtration model; you specify only C_f to define a static filtration model.

The porosity associated with the slurry is defined in terms of the slurry concentration as $\varepsilon_s = 1 - C_v$. The rate of growth of the thickness of the filter cake layer is assumed to be:

$$\frac{dh}{dt} = b\frac{\kappa}{\mu}\frac{\partial p}{\partial x}$$

where $b = \frac{1-\varepsilon_s}{\varepsilon_s-\varepsilon_h}$ is the specific filter cake volume, p represents the fluid pressure in the filter cake, and x represents the coordinate in the thickness direction of the filter cake. You may optionally specify an initial value of the filter cake thickness, h_0. The above equation is integrated in time to obtain the thickness of the filter cake layer, $h(t)$, as a function of time. In the incompressible case, the specific filter cake volume simplifies to:

$$b = \frac{C_v}{1-C_v-C_f\varepsilon^*}$$

The permeability of the filter cake is computed as:

$$\kappa = C_d\frac{\varepsilon_h}{1-\varepsilon_h}$$

where C_d is a permeability factor that you can specify directly. If you do not specify the permeability factor, it is computed internally by Abaqus as:

$$C_d = \frac{\varepsilon_0^2 D_p^2}{150(1-\varepsilon_0)}$$

where D_p is the particle diameter. In the case of an incompressible filter cake (that is, $\alpha = 0$), the expression for permeability becomes:

$$C_d = \frac{\varepsilon_0^3 D_p^2}{150(1-\varepsilon_0)^2}$$

which is very similar in form to the well-known Kozeny-Carman equation describing the permeability for a fluid flowing through a packed bed of solids.

The flux of the normal flow across the cohesive surfaces is:

$$q_t = -\frac{\kappa}{\mu}\frac{(p_f - p_p)}{h(t)}\Omega$$

where κ and μ represent the permeability of the filter cake and the filtrate viscosity, respectively, $h(t)$ is the time-dependent thickness of the filter cake layer, and $\Omega\,(\leq 1.0)$ is a numerical factor that accounts for the compressibility of the filter cake layer. For the dynamic filtration model, Abaqus computes the factor Ω based on α, while for the static filtration model, $\Omega = 1.0$.

Input File Usage: Use the following option to define the physics-based leak-off model:

**FLUID LEAKOFF*, SLURRY

Abaqus/CAE Usage: Definition of physics-based leak-off model or corresponding user-subroutine are not supported in Abaqus/CAE.

User Subroutine-Based Leak-off Behavior

You can use user subroutine *UFLUIDLEAKOFF* to define a physics-based model of your choice. In this case, you define the fluid flux across the top and the bottom crack surfaces and their derivatives with respect to the degrees of freedom directly in user subroutine *UFLUIDLEAKOFF*.

Input File Usage: Use the following option to define more complex leak-off behavior in user subroutine *UFLUIDLEAKOFF*:

**FLUID LEAKOFF*, SLURRY, USER

Abaqus/CAE Usage: Defining a user-specified leak-off model is not supported in Abaqus/CAE.

Offset of Cracked Element Surfaces

A crack opening offset for contact of cracked element surfaces (see *Defining Contact of Cracked Element Surfaces Using a Small-Sliding Formulation*) refers to a value of the separation in the normal direction, $\delta_n = \delta_n^{off}$, below which the resistance to additional closure $\left(\delta_n > 0,\ \dot{\delta}_n < 0\right)$ of the cracked element surfaces increases significantly. This capability is useful in preventing the closure of a fracture when changes in external loading conditions might otherwise result in closure. A practical application of this capability occurs during hydraulic fracturing with slurry, where solid proppant particles (the solid part of the slurry) help prop open the fracture when external hydraulic pressure loads are removed.

In an enriched element, an offset for contact between the cracked surfaces is specified using user subroutine *UXFEMCRACK*.

Thermal Effect with Slurry Transport

To model heat transfer associated with slurry transport within a crack, you must define the relevant thermal properties. Optionally, the thermal properties can be defined as functions of the slurry concentration. In particular, the thermal conductivity can be defined as a function of the slurry concentration directly in a tabular form. However, the specific heat of the slurry is computed automatically based on the specific heats of the carrying fluid and the proppant particles, respectively, using a rule of mixtures.

Input File Usage: Use the following option to specify the heat capacity of the carrier fluid and the proppant particles. The heat capacities can be either constants or functions of temperature and field variables.

**SPECIFIC HEAT*, PORE FLUID, SLURRY

Use the following option to specify the thermal conductivity of the slurry. The conductivity of the slurry can be either a constant or a function of slurry concentration, temperature, and field variables.

**CONDUCTIVITY*, PORE FLUID, SLURRY

Abaqus/CAE Usage: Defining the heat capability and conductivity properties of the slurry fluid is not supported in Abaqus/CAE.

Limits on Slurry Concentration

Depending upon the nature of packing of the proppant particles (assumed to be spherical), there will be a theoretical upper limit on the slurry concentration. By default, Abaqus assumes this upper limit to be 0.67. The lower limit of the slurry concentration is assumed to be 0. The strongly advective nature of the slurry transport problem can lead to oscillations in the solution; therefore, these theoretical limits might be violated numerically in some cases. By default, Abaqus uses a box-bound constraint approach to enforce limits on the nodal degree of freedoms representing the slurry concentration. In addition, the use of a Pertov Galerkin scheme to formulate the slurry transport continuity equation helps alleviate such numerical oscillations to a large extent but does not completely eliminate them. As discussed earlier (see *Stability of Tangential Flow*), the use of artificial diffusivity is recommended for most applications.

You can specify nondefault upper and lower limits on the slurry concentration.

Input File Usage: Use the following option to specify nondefault upper and the lower limits on the slurry concentration:

In a geostatic procedure:

**GEOSTATIC*, SLURRY MIN=*min_value*,SLURRY MAX=*max_value*

In a soils consolidation procedure:

**SOILS*, CONSOLIDATION, SLURRY MIN=*min_value*,SLURRY MAX=*max_value*

Abaqus/CAE Usage: Specifying nondefault upper and the lower limits on the slurry concentration is not supported in Abaqus/CAE.

Simulating Flow of Clear Fluid

Some hydraulic fracturing applications involve the use of clear fluid to initiate fracture, followed by the use of a slurry. To model such applications, you can use multiple steps in the analysis, and specify that Abaqus does not solve for slurry concentration during the step in which a clear fluid is used.

Input File Usage: Use the following option to define a flow of clear fluid:

In a geostatic procedure:

**GEOSTATIC*, SLURRY FLOW=NO

In a soils consolidation procedure:

**SOILS*, CONSOLIDATION, SLURRY FLOW=NO

Abaqus/CAE Usage: Definition of a flow of clear fluid is not supported in Abaqus/CAE

Output

The following surface output variables are available only when the slurry fluid flow is enabled within the cracked enriched element surfaces:

FLVF When multiple fluids are defined, the components of FLVF represent the volume fraction of each fluid type. In particular, FLVF_*i* refers to the volume fraction of fluid type *i*.

SLURRYVF Volumetric concentration of proppant particles in slurry within the cracked surfaces in the enriched element.

SLURRYAF Volume of proppant particles in slurry per unit area within the cracked surfaces in the enriched element.

THKFTCK Filter cake thickness within the cracked surfaces in the enriched element.

References

- Outmans, H. D., “Mechanics of Static and Dynamic Filtration In the Borehole,” *Society of Petroleum Engineers Journal*, vol. 3, pp. 236–244, 1963.

Reduced Basis Generation

Products: Abaqus/Standard

Overview

Reduced basis generation:

- reads deformation fields and matrices that have been generated by preceding nonlinear procedures and, optionally, modes that have been generated by preceding natural frequency extraction procedures;
- computes a set of basis vectors that reproduce the nonlinear deformation fields and the natural frequency modes when superimposed;
- projects the operator data from the preceding nonlinear procedures onto the reduced basis; and
- writes the reduced basis and the projected operators to a SIM document and optionally to a nonlinear flexible body input file for use in the Simpack flexible body dynamics solver.

References:

- *Generating Matrices from Nonlinear Procedures*
- **FLEXIBLE BODY*
- **OPERATOR OUTPUT*
- **REDUCED BASIS GENERATE*
- **SELECT EIGENMODES*

Introduction

The term reduced basis refers to a set of basis vectors that might represent a set of nonlinear deformation fields from preceding procedures when superimposed with appropriate scaling factors. Such deformation fields might have been computed, for example, from ramping up a load that acts on a finite element model in a nonlinear static analysis. Optionally, the reduced basis can also represent a set of modes that might have been computed from one or more natural frequency extraction procedures with varying predeformation. A reduced basis can be used, for example, with hyperreduction analysis techniques or to set up a nonlinear flexible body for use in the Simpack flexible body dynamics solver. Compared to the original finite element model, such analysis techniques allow for a reduction of the computation time by several orders of magnitude for nonlinear static or dynamic scenarios.

Generating Reduced Bases

As input data, the reduced basis generation procedure collects nonlinear deformation fields and matrices from the generated SIM documents. It then computes an initial set of orthonormal basis vectors, utilizing a singular value decomposition of the nonlinear deformation fields. The singular value that is associated with every basis vector determines if the basis vector is retained: Starting with the smallest singular value, basis vectors are neglected until the sum of the associated singular values exceeds a specified fraction of the overall sum. By default, this fraction is 10^{-4}.

Optionally, you can use the reduced basis generation procedure to collect the modes that have been computed by preceding natural frequency extraction procedures. It then computes additional basis vectors by means of a second singular value decomposition. By default, basis vectors that are associated with the same fraction of the

sum of singular values are neglected as for the first singular value decomposition. The remaining basis vectors are orthonormalized with respect to the first set of basis vectors. The first set of basis vectors and the resulting residual vectors are then combined. If the SIM documents from the preceding nonlinear procedures contain the mass matrix of the undeformed finite element model, all basis vectors are orthonormalized with respect to that mass matrix.

Finally, the reduced basis generation procedure projects all the matrices that are contained in the binary SIM files from the preceding nonlinear procedures onto the reduced basis.Abaqus/Standard stores the reduced basis and the projected operators in a new SIM document. By default, this file is named *jobname*_RB*n*.sim, where *jobname* is the name of the input file or analysis job and *n* is the number of the Abaqus step that generates the reduced basis.

Input File Usage: Use the following options to specify the reduced basis generation:

```
** Nonlinear procedure with the operator output option
*STEP, NLGEOM=YES
*STATIC

*OPERATOR OUTPUT, SOLUTION,
*END STEP
** Reduced basis generation procedure
*STEP
*REDUCED BASIS GENERATE
fraction of singular value sum to neglect
*END STEP
```

Use the following options to consider natural frequency modes for the reduced basis generation:

```
** Natural frequency extraction (undeformed finite element model)
*STEP, PERTURBATION
*FREQUENCY

*END STEP
** Nonlinear procedure with the operator output option (first deformation step)
*STEP, NLGEOM=YES
*STATIC

*OPERATOR OUTPUT, NUMBER INTERVAL=1, SOLUTION,
*END STEP
** Natural frequency extraction (predeformed finite element model)

** Further alternating nonlinear procedures and natural frequency extractions

** Reduced basis generation procedure
*STEP
*REDUCED BASIS GENERATE
fractions of the sums of singular values to neglect (nonlinear deformation fields,
natural frequency modes)
*SELECT EIGENMODES
modes to be considered from every one of the preceding natural frequency extractions
*END STEP
```

Use the following option to specify the name of the file that the reduced basis generation procedure creates:

*REDUCED BASIS GENERATE, NAME=*name*

Generating Nonlinear Flexible Bodies

Optionally, you can use the reduced basis generation procedure to generate a flexible body input `.fbi` file for use in the Simpack flexible body dynamics solver. In Simpack, you can assign the `.fbi` file directly to a nonlinear flexible body.

Input File Usage: Use the following options to generate a nonlinear flexible body input file:

```
** Reduced basis generation procedure
*STEP
*REDUCED BASIS GENERATE

*FLEXIBLE BODY, TYPE=SIMPACK
*END STEP
```

Requirements for Reduced Basis Generation

To achieve good performance of downstream simulations in Simpack in terms of result accuracy and computation times, the reduced basis generation should fulfill the following requirements:

- In the assembled system, the nonlinear flexible body has one characteristic large and nonlinear deformation shape. Usually, this deformation shape corresponds to a compliant loading direction. For example, this might be the bending of a wind turbine rotor blade in the flap-wise direction or the torsion of a twist beam in the rear axle of a car assembly during a lane change maneuver at high speeds.
- Abaqus generates nonlinear deformation fields and matrices for the compliant loading direction by one or more nonlinear static procedures. When you define the loads and boundary conditions for this procedure, it might be necessary to constrain the rigid motion of the finite element model by adding appropriate connector elements and constraints. In such cases, exclude the additional connector elements from the operator output by specifying appropriate node and element sets that include only the bare finite element model. In any case, you should exclude the boundary conditions from the operator output.
- The load and solution vectors, the mass matrix, the stiffness matrix, and the free body response vector are generated and output from the nonlinear static procedures at various deformation levels. Because only the mass matrix for the undeformed finite element model is required, you need to output it only from the procedure that starts from or ends with the undeformed model.
- Natural frequency modes are computed for a sufficient number of deformation levels. This is achieved by ramping up the load in the compliant direction of the model step-by-step with multiple subsequent nonlinear static procedures. Between every two such procedures, perform a natural frequency extraction. In addition, perform a natural frequency extraction before the first and after the last nonlinear static procedure. Typically, you need approximately 40 to 60 alternating nonlinear static procedures and natural frequency extractions. The operator output is required only for the last increment in every nonlinear static procedure.
- The fractions of the sum of singular values to neglect are configured such that fewer than 10 basis vectors are needed to represent the nonlinear deformation fields and approximately 10 to 100 additional basis vectors are needed to represent the natural frequency modes of the undeformed and the deformed finite element model are retained. Typical values for such fractions lie in a range between 10^{-3} and 10^{-5}.

Input File Usage: Use the following options to exclude boundary conditions and other auxiliary elements that serve to constrain rigid motion during operator generation:

**OPERATOR OUTPUT*, NSET=*name of set that contains all nodes of the bare finite element model*, ELSET=*name of set that contains all elements of the bare finite element model*, BOUNDARY CONDITIONS=NO, ...

Use the following options to generate load, solution, mass, stiffness, and free body response operators:

**OPERATOR OUTPUT*, LOAD, SOLUTION, MASS, STIFFNESS, FREE BODY RESPONSE,

Use the following options to output operators only during the last increment when using alternating nonlinear static procedures and natural frequency extractions:

**OPERATOR OUTPUT*, NUMBER INTERVAL=1,

Use the following options to specify the fractions of the sum of singular values to neglect during reduced basis generation:

**REDUCED BASIS GENERATE*
fraction of singular value sum to neglect (solution vectors), fraction of singular value sum to neglect (eigenmode shapes)

Generating the Reduced Basis Using the Restart Capability

You can generate the reduced basis in the same job that contains the nonlinear procedures and the natural frequency extractions. Alternatively, you can use the restart capability. In that case, the job that generates the reduced basis must restart from the job that contains the last nonlinear procedure or natural frequency extraction. To consider natural frequency modes, request the generation of data for the restart analysis in all the natural frequency extraction procedures.

Input File Usage: To consider natural frequency modes for the reduced basis generation, use the following options to generate the data for the restart analysis in the first job (and in all subsequent jobs):

** Natural frequency extraction (undeformed finite element model)
**STEP*, PERTURBATION
**FREQUENCY*

**RESTART*, WRITE
**END STEP*
** Nonlinear static procedure (first deformation step)
**STEP*, NLGEOM=YES
**STATIC*

**OPERATOR OUTPUT*, NUMBER INTERVAL=1, SOLUTION,
**END STEP*
** Natural frequency extraction (predeformed finite element model)

** Further alternating nonlinear static procedures and natural frequency extractions

Specify all alternating nonlinear procedures and natural frequency extractions in the first job, or split them into a number of subsequent jobs using the restart capability,

as appropriate. Use the following options to generate the reduced basis in the job that restarts from the last of the preceding jobs:

```
*RESTART, READ
** Reduced basis generation procedure
*STEP
*REDUCED BASIS GENERATE

*END STEP
```

Special-Purpose Techniques

In this section:

- *Inertia Relief*
- *Element and Contact Pair Removal and Reactivation*
- *Progressive Element Activation*
- *Introducing a Geometric Imperfection into a Model*
- *Fracture Mechanics*
- *Surface-Based Fluid Modeling*
- *Mass Scaling*
- *Selective Subcycling*
- *Steady-State Detection*
- *CZone Analysis*

Inertia Relief

Products: Abaqus/Standard Abaqus/CAE

Overview

Inertia relief:

- involves balancing externally applied forces on a free or partially constrained body with loads derived from constant rigid body accelerations;
- requires material density or mass and/or rotary inertia values to be specified for computing inertia relief loads;
- can be performed for static, dynamic, and buckling analyses in Abaqus/Standard;
- varies the inertia relief loading with the applied loading in static analysis;
- applies inertia relief load corresponding to the static preload in dynamic analysis;
- can be used to balance applied perturbation loads when used with buckling analysis;
- uses rigid body accelerations consistent with the specified boundary conditions to compute the inertia relief loads;
- can be geometrically linear or nonlinear;
- may require the use of the unsymmetric solver if there are large inertia relief moments in a geometrically nonlinear analysis;
- is an inexpensive alternative to doing a full dynamic free body analysis when applied loads vary slowly compared to the eigenfrequencies of the body; and
- can be used with multiple load cases.

References:

- *Distributed Loads*
- *Defining an Analysis*
- **INERTIA RELIEF*
- *Defining an inertia relief load*

Typical Applications

Inertia relief loading can be applied in static (*Static Stress Analysis*), dynamic (*Implicit Dynamic Analysis Using Direct Integration*), and eigenvalue buckling prediction steps (*Eigenvalue Buckling Prediction*).

In a static step the inertia relief loading varies with the applied external loading. An example of using an inertia relief load is modeling a rocket undergoing constant or slowly varying acceleration during lift-off (i.e., a free body subjected to a constant or slowly varying external force) with a static analysis procedure. The inertia forces experienced by the body are included in the static solution through inertia relief loading that balances the external loading.

In a dynamic step the inertia relief loading is calculated based on the static preload and is held constant during the step. The following is an example of using an inertia relief load in a dynamic analysis procedure: Consider a free body submerged in water and subjected to shock wave loading due to an explosion. A dynamic analysis is needed to compute the transient solution. If it is known that initially the body is stationary under gravity and hydrostatic pressure from the fluid, the gravity load should exactly balance the buoyancy force. However, if the finite element model does not include all the mass existing in the body (for example, ballast), without additional

loading, the body would accelerate due to out-of-balance external forces. Applying inertia relief loading exactly balances these unbalanced external loads, placing the body in static equilibrium. The dynamic analysis then provides the transient response of the body to the shock wave loading as deformation of the body relative to its static equilibrium position.

In a buckling analysis the inertia relief load can be applied in the static preload step, in the eigenvalue buckling prediction step, or in both steps. In the eigenvalue buckling prediction step the inertia relief load is calculated based on the perturbation loads. Consider the static analysis rocket example. If we use inertia relief in a buckling analysis of the rocket with the rocket thrust as the perturbation load, we can predict the critical thrust that causes the rocket to buckle.

Basic Formulation

In inertia relief the total response, $\{u_t\}$, of the body is written as a combination of a rigid body response due to rigid body motion of a reference point, $\{u_b\}$, and a relative response, $\{u\}$:

$$\{u_t\} = \{u\} + \{u_b\},$$

with corresponding expressions for velocities and accelerations. The reference point is the center of mass except when you must specify the reference point. Then, the finite element approximation to the dynamic equilibrium equation becomes

$$[M]\ \{\ddot{u}\} + [M]\ \{\ddot{u}_b\} + \{I\} = \{P\},$$

where $[M]$ is the mass matrix, $\{I\}$ is the internal force vector, and $\{P\}$ is the external force vector. The response of interest in a static analysis involving inertia relief is the rigid body response corresponding to the dynamic motion of the reference point and the static response relative to the rigid body motion. Hence, the relative acceleration term $[M]\ \{\ddot{u}\}$ drops from the equilibrium equation.

The rigid body response can be expressed in terms of the acceleration of the reference point, $\ddot{z}_j$, and rigid body mode vectors, $\{T\}_j$, $j = 1, 2, \ldots, 6$ (in three dimensions):

$$\{\ddot{u}_b\} = \sum_{j=1}^{6} \{T\}_j \ddot{z}_j.$$

By definition, $\{T\}_j$ represents the acceleration vector corresponding to a unit imposed acceleration (displacement or rotation) in the j-direction at the reference point. For example, at a node with the usual three displacements and three rotations $\{T\}_j$ is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & (z-z_0) & -(y-y_0) \\ 0 & 1 & 0 & -(z-z_0) & 0 & (x-x_0) \\ 0 & 0 & 1 & (y-y_0) & -(x-x_0) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \\ \hat{e}_4 \\ \hat{e}_5 \\ \hat{e}_6 \end{Bmatrix},$$

where $\hat{e}_j$ is unity; all other $\hat{e}_i$ are zero; x, y, and z are the coordinates of the node; and x_0, y_0, and z_0 represent the coordinates of the reference point that is the center of rotation. If the system undergoes finite changes in geometry, $\ddot{z}_j$ and $\{T\}_j$ will both be functions of time.

Projecting the dynamic equilibrium equation onto the rigid body modes, we have

$$\sum_{j=1}^{6} m_{ij}\ddot{z}_j = \{T\}_i^T \{P\},$$

where $m_{ij} = \{T\}_i^T [M] \{T\}_j$ is the "rigid body inertia" and $\ddot{z}_j$ is the rigid body acceleration associated with the rigid body mode *j*. The actual number of rigid body modes will be less than 6 in the presence of symmetry planes as well as for two-dimensional and axisymmetric analyses. Thus, the rigid body response can be evaluated directly from the external loads.

The relative response of the body can be obtained by solving the equilibrium equation with the known inertial term $[M] \{\ddot{u}_b\}$ moved to the right hand side; that is, applied as a body force. The static equilibrium equation then becomes

$$\{I\} = \{P\} + \{P^{ir}\},$$

where $\{P^{ir}\} = -[M] \sum_{j=1}^{6} \{T\}_j \ddot{z}_j$.

In a dynamic analysis involving inertia relief the rigid body mode vectors $\{T\}_j^0$ are calculated in the configuration at the start of the dynamic analysis, and the reference point accelerations $\ddot{z}_j^0$ are calculated to balance the static preloads in this configuration. The relative acceleration term is not dropped, so the dynamic equilibrium equation becomes

$$[M] \{\ddot{u}\} + \{I\} = \{P\} + \{P^{ir}\},$$

where $\{P^{ir}\} = -[M] \sum_{j=1}^{6} \{T\}_j^0 \ddot{z}_j^0$. In a geometrically nonlinear analysis the rigid body mode vectors are recomputed during the analysis using the current configuration but the reference point accelerations are kept constant. This keeps the total magnitude of inertia relief loads constant during the analysis but allows the loads to be proportional to the spatial mass distribution, which changes with geometry.

Input File Usage: **INERTIA RELIEF*

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**

Inertia Relief Loading Directions

By default, all rigid body motion directions in a model can be loaded by inertia relief loading (in this discussion we use the word "direction" to mean any rigid body translation or rotation). In models with symmetry planes or models that are allowed to move freely in only specific directions, the free directions for which inertia relief loading is applied can be specified. For example, in a three-dimensional analysis with one symmetry plane only three free directions exist—two translations and one rotation. Add an additional symmetry plane and only one free translation remains. A cylinder-piston arrangement is an example where the only free direction considered is motion along the cylinder's axis. In these situations you specify the free directions that are loaded by inertia relief loading by indicating the degrees of freedom.

The case of two free rotation directions is not permitted. For cyclic symmetric models with inertia relief only translation in the *Z*-direction and rotation about the *Z*-direction are considered for computing inertia relief loading.

Input File Usage: **INERTIA RELIEF*
integer list of global degrees of freedom identifying the free directions

For example, the list 1, 3, 5 implies that translations in the *X*- and *Z*-directions and rotation about the *Y*-axis are free directions.

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: toggle on the degrees of freedom to define the **Free Directions** (the degrees of freedom displayed are dependent on the modeling space)

Defining the Free Directions in a Local Coordinate System

If the free directions are not global directions, an orientation can be used to define the local coordinate system to which the integer list of degree of freedom identifiers refers.

Input File Usage: **INERTIA RELIEF*, ORIENTATION=*orientation_name*
integer list of local degrees of freedom identifying the free directions

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: click **Edit**, and choose a local **CSYS**

Defining Free Direction Combinations That Require a User-Specified Reference Point

Not all user-chosen combinations of free directions admit unconstrained rigid body motion; that is, there are certain combinations of free directions for which an additional point is required to define the rigid body motion vectors. For example, in three dimensions the choice 4, 5, 6 corresponds to free rotations about a fixed point. The fixed point must be given to define the rigid body motion vectors. In other examples the free directions include rotation about a fixed axis. Consider a turbine blade rotating about its axis, as shown in *Figure 1*.

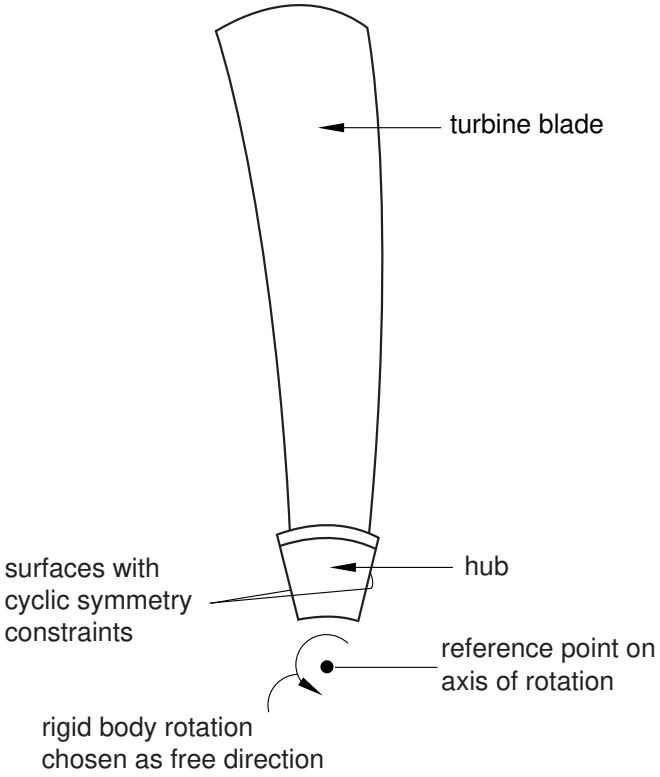


Figure 1: Inertia relief for a turbine blade with rotation about the axis as the only free direction.

To find the angular acceleration of the blade as it rotates under an applied force couple or moment, you should specify the coordinates of the point on the shaft about which the blade is rotating. The free direction combinations for which you must specify a reference point are given in *Table 1*.

Table 1: Free direction combinations requiring a reference point.

Degree of freedom identifiers defining free directions	Reference point definition		
	Fixed rotation point	Point on rotation axis	Point on symmetry line
4, 5, 6	√		
1, 4, 5, 6	√		
2, 4, 5, 6	√		
3, 4, 5, 6	√		
1, 2, 4, 5, 6	√		
1, 3, 4, 5, 6	√		
2, 3, 4, 5, 6	√		
4		√	
5		√	

Degree of freedom identifiers defining free directions	Reference point definition		
	Fixed rotation point	Point on rotation axis	Point on symmetry line
6		√	
2, 4		√	
3, 4		√	
1, 5		√	
3, 5		√	
1, 6		√	
2, 6		√	
1, 2, 4		√	
1, 2, 5		√	
1, 3, 4		√	
1, 3, 6		√	
2, 3, 5		√	
2, 3, 6		√	
1, 4			√
2, 5			√
3, 6			√

Input File Usage: **INERTIA RELIEF*, ORIENTATION=*orientation_name*
integer list of local degrees of freedom identifying the free directions
X, Y, Z coordinates of the reference point for defining the rigid body vectors

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: toggle on **Global position of reference point**, and enter the **X**, **Y**, and (if available) **Z** coordinates

Initial Conditions

Initial conditions can be specified in the same way as in static and dynamic analyses without inertia relief loads. If inertia relief is used in the first step in the analysis, these initial conditions form the base state of the body. See *Initial Conditions*.

Boundary Conditions

Boundary conditions are specified in the same way as in analyses without inertia relief loads (see *Boundary Conditions*). In theory, a statically determinate set of restraints is needed when inertia relief is used in a static step. By "statically determinate" we mean a set of restraints that restrain all rigid body modes but no deformation modes. Such a set provides a unique displacement solution and ensures that the inertia relief loading exactly balances the user-specified external loading: zero reaction forces with no rigid body motion of the center of mass. *Table 2* summarizes the restraint requirements for various cases.

Table 2: Necessary and sufficient statically determinate restraints.

Problem dimensionality	Free directions	Number of required restraints
2D	2 Translations and 1 Rotation	3
Axisymmetric	1 Translation	1
Axisymmetric with twist	1 Translation and 1 Rotation	2
3D	3 Translations and 3 Rotations	6

However, it is not necessary for the user to explicitly specify boundary conditions (*Boundary Conditions*) with inertia relief except in the case of buckling analysis. If no boundary conditions or insufficient boundary conditions are specified, a warning message will be issued and boundary conditions necessary to restrain the rigid body modes will be imposed internally at the point in the model that corresponds to the original location of the reference point. On the other hand, if too many boundary conditions are specified in certain directions, a warning message will be issued to indicate that the reaction forces may be nonzero at the nodes with overspecified boundary conditions. If there are insufficient boundary conditions in certain directions and too many boundary conditions in other directions, the problem will be treated as a combination of these cases.

If a model has no boundary conditions or insufficient boundary conditions, a particular number of numerical singularity warnings can be issued during each equilibrium iteration in the analysis. The displacement solution is postprocessed to remove unconstrained rigid body motion. However, the number of numerical singularities should not exceed the number of unconstrained rigid body modes; any extra numerical singularity messages may indicate other problems.

Similarly, a model with no boundary conditions or insufficient boundary conditions may produce negative eigenvalue messages. If the number of negative eigenvalues at each equilibrium iteration in the analysis does not exceed the maximum reasonable number of numerical singularities associated with the boundary conditions for inertia relief, the results can be trusted, but extra negative eigenvalues may indicate other problems.

If a model contains symmetry planes or is constrained to move freely in specific directions, inertia relief loading should be applied only in those free directions. No boundary conditions should be specified in the free directions; however, sufficient boundary conditions must be specified in the other directions. Any boundary conditions that violate the above requirements will be flagged as an error. An error will also be issued if the combination of free directions includes only two free rotations or if a reference point is required but not specified.

In a buckling analysis, proper boundary conditions are important for getting the correct mode shape. Sufficient boundary conditions must be specified when inertia relief loading is applied in such an analysis. See *Eigenvalue Buckling Prediction* for details on how to apply boundary conditions in a buckling analysis.

Loads

An analysis that uses inertia relief can include concentrated nodal forces at displacement degrees of freedom (1–6), distributed pressure forces or body forces, and user-defined loading.

Inertia relief loads are used to balance the external loads. They are computed and applied when inertia relief is included in the step definition. The rules for propagating load definitions between steps hold for inertia relief

loads. See *About Loads*. The inertia relief loads will not be propagated to steps where inertia relief is not valid for the specified procedure.

If there are large inertia relief moments in a geometrically nonlinear analysis, their contribution to the stiffness matrix may be unsymmetric. In such cases unsymmetric equation solution may improve the computational efficiency (see *Defining an Analysis*).

Computing Inertia Relief Loads

The nodal force vector corresponding to the inertia relief loads is calculated as follows. The applied loads are projected onto the rigid body modes, $\{T\}_j$. These force and moment components (six components in three dimensions) are used with the "rigid body inertia" to solve for the rigid body accelerations, $\ddot{z}_j$. Only the rigid body acceleration components corresponding to the inertia relief loading directions are nonzero. The nodal force vector is calculated using the assembled mass matrix $[M]$ as

$$\{P^{ir}\} = -[M] \sum_{j=1}^{6} \{T\}_j \ddot{z}_j.$$

Fixed Inertia Relief Loads

You can specify that the inertia relief loads should be held fixed in magnitude and direction at the values calculated at the end of the previous step.

Input File Usage: **INERTIA RELIEF*, FIXED

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: **Method: Fix at current loading**

Removing Inertia Relief Loads

You can specify that the inertia relief loads that were applied in the previous general analysis step should be removed in the current step.

Input File Usage: **INERTIA RELIEF*, REMOVE

Abaqus/CAE Usage: Load module: **Load Manager**: **Deactivate**

Predefined Fields

User-defined field variables can be specified in the same way as in static and dynamic analyses without inertia relief loads. See *Predefined Fields*.

Material Options

Any of the mechanical constitutive models that are available in Abaqus/Standard for use in static, dynamic, or buckling analyses can be used with inertia relief (see *Abaqus Materials Guide* for details on the material models available in Abaqus/Standard). Since inertia relief loading is calculated using the inertia properties of the model, the density must be specified (see *Density*) to define the model's inertia properties.

Elements

Most of the stress/displacement elements that are available in Abaqus/Standard for use in static, dynamic, and buckling analyses (including mass and rotary inertia elements and user elements) can be used. A warning will be issued when the model contains elements that do not have associated mass or inertia (for example, hydrostatic fluid elements and pore pressure elements). An error will be issued if the model contains elements that do not allow finite boundaries (for example, infinite elements and elastic element foundations). Although five degree of freedom shell elements can be used in a step with inertia relief loads, they may cause convergence difficulties if the model has no boundary conditions or insufficient boundary conditions. To improve convergence, these elements should be replaced with other conventional shell elements.

In the case of a substructure you must generate a reduced mass matrix for the substructure (see *Generating a Reduced Structural Damping Matrix for a Substructure*). The reduced mass matrix is included in the global mass matrix of the entire model to compute rigid body accelerations and inertia relief loads. Inertia relief can be used only with substructures in a geometrically linear analysis. An error message is issued if inertia relief is used with substructures in a geometrically nonlinear analysis.

Output

In addition to the usual output variables available in Abaqus/Standard (see *Abaqus/Standard Output Variable Identifiers*), the following variables are provided specifically for inertia relief:

Variables for the entire model:

IRX
Current coordinates of the reference point.

IRX*n*
Coordinate *n* of the reference point ($n = 1, 2, 3$).

IRA
Equivalent rigid body acceleration components.

IRA*n*
Component *n* of the equivalent rigid body acceleration ($n = 1, 2, 3$).

IRAR*n*
Component *n* of the equivalent rigid body angular acceleration with respect to the reference point ($n = 1, 2, 3$).

IRF
Inertia relief load corresponding to the equivalent rigid body acceleration.

IRF*n*
Component *n* of the inertia relief load corresponding to the equivalent rigid body acceleration ($n = 1, 2, 3$).

IRM*n*
Component *n* of the inertia relief moment corresponding to the equivalent rigid body angular acceleration with respect to the reference point ($n = 1, 2, 3$).

IRRI
Rotary inertia about the reference point.

IRRI*ij*
ij-component of the rotary inertia about the reference point ($i \le j \le 3$).

IRMASS
Whole model mass.

For most cases inertia relief loads correspond to the product of "rigid body inertia" and the equivalent rigid body acceleration vector. However, when only a few rigid body directions are chosen as free directions for inertia relief, inertia relief loads are computed in all rigid body directions for output purposes, but equivalent rigid body accelerations are computed in only the free directions with the equivalent rigid body angular accelerations computed from the diagonal entries of the "rigid body inertia."

Limitations

You need to be aware of limitations that may be encountered in analyses with inertia relief loads.

Internal Boundary Conditions and Convergence in Geometrically Linear and Nonlinear Analysis

In a model containing internal boundary conditions that generate unbalanced internal forces or moments, such as is possible from certain elements (for example, SPRING1, DASHPOT1, SPRING2, DASHPOT2, or GAPUNI elements) or kinematic constraints (for example, coupling constraints, linear constraint equations, multi-point constraints, or surface-based tie constraints), inertia relief loads will not balance these internal forces or moments. If the model contains sufficient boundary conditions, these internal forces or moments will appear as nonzero reaction forces or moments. If the model does not contain sufficient boundary conditions, these internal forces or moments will appear as unconverged residual fluxes in the message file for geometrically linear as well as nonlinear analyses. The model should be treated as having internal boundary conditions, with the unconverged residuals representing the reaction forces or moments needed to impose the internal boundary conditions. Ideally, the internal boundary conditions should be removed or sufficient boundary conditions should be added to the model.

Unconnected Regions and Analyses with Contact

Inertia relief is not supported for models consisting of multiple unconnected regions, even if contact is defined between them. An exception is when tied contact is defined between the regions. In this case it is the user's responsibility to ensure that different parts are tied in such a way that no rigid body motion is possible between them.

In addition, models involving contact with inertia relief loads may show poor convergence or fail to converge in cases when the surfaces are not in contact or when contact stabilization is used.

Mass and Stiffness Defined Using Matrices

Mass and stiffness cannot be defined using matrices in analyses with inertia relief loads.

Assembly Loads

An analysis with inertia relief and assembly loads (see *Prescribed Assembly Loads*) may experience poor convergence or no convergence if the model is not properly constrained by boundary conditions.

Input File Template

```
*HEADING
…
*DENSITY
Data line to specify material density
*BOUNDARY
Data lines to specify zero-valued boundary conditions
**
*STEP (, NLGEOM) (, PERTURBATION)
Use the NLGEOM parameter to include nonlinear geometric effects;
it will remain active in all subsequent steps.
```

```
*STATIC (or *DYNAMIC)
…
*CLOAD and/or *DLOAD
Data lines to specify loads
*INERTIA RELIEF, ORIENTATION=orientation_name
Data lines to specify global (or local, if the ORIENTATION parameter is used) degrees
of freedom that define free directions and to provide coordinates of a reference point
*END STEP
**
*STEP
*STATIC(or *DYNAMIC)
…
*INERTIA RELIEF, FIXED or REMOVE
Include the FIXED parameter to keep inertia relief loads fixed at their current
values from the beginning of the step; include the REMOVE parameter to
remove inertia relief loads from the beginning of the step.
*END STEP
```

Element and Contact Pair Removal and Reactivation

Products: Abaqus/Standard Abaqus/CAE

References:

- *Removing and Reactivating Contact Pairs*
- **MODEL CHANGE*
- *Defining a model change interaction*

Overview

Element and contact pair removal/reactivation:

- can be used to simulate removal of part of the model, either temporarily or for the remainder of the analysis;
- allows reactivation of elements strain-free or with strain;
- can be used to save computational time when a contact pair is not needed;
- can be used only in general analysis steps; and
- can be used in a restart analysis only if it was used or activated in the original analysis.

Removing Elements

You can remove specified elements from the model in a general analysis step. Just prior to the removal step, Abaqus/Standard stores the forces/fluxes that the region to be removed is exerting on the remaining part of the model at the nodes on the boundary between them. These forces are ramped down to zero during the removal step; therefore, the effect of the removed region on the rest of the model is completely absent only at the end of the removal step. The forces are ramped down gradually to ensure that element removal has a smooth effect on the model.

No further element calculations are performed for elements being removed, starting from the beginning of the step in which they are removed. The removed elements remain inactive in subsequent steps unless you reactivate them as described below.

Input File Usage: Use the following option to remove elements from the model:

**MODEL CHANGE*, TYPE=ELEMENT, REMOVE

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Deactivated in this step**

Removing Elements in Transient Procedures

Care must be taken in removing elements in transient procedures. The nodal flux that the removed elements apply at the boundary with the rest of the model is ramped down over the step. In transient heat transfer, fully coupled temperature-displacement, or fully coupled thermal-electrical-structural analysis if the fluxes are high and the step is long, this ramping down may have the effect of cooling down or heating up the rest of the body. In dynamic analysis if the forces are high and the step is long, kinetic energy can be imparted to the remaining

portion of the model. This problem can be avoided by removing the elements in a very short transient step prior to the rest of the analysis. This step can be done in a single increment.

Reactivating Stress/Displacement Elements

Two distinct types of reactivation are provided for stress/displacement elements (including substructures): strain-free reactivation and reactivation with strain. Strain-free reactivation resets the initial configuration; reactivation with strain does not.

Although elements cannot be created within an analysis, a similar effect can be achieved by creating elements in the model definition, removing them in the first step, and subsequently reactivating them.

Strain-Free Reactivation

When stress/displacement elements are reactivated in a strain-free state, they become fully active immediately at the moment of reactivation (the start of the step in which they are reactivated). They are reset to an "annealed" state (zero stress, strain, plastic strain, etc.) in the configuration in which they lie at the start of the reactivation step. This configuration depends on whether a small- or large-displacement analysis is being conducted. Alternatively, reactivation in a nonvirgin state can be specified, as described below.

Since these elements are reactivated in a virgin state (i.e., with zero stress), they exert zero nodal forces on the rest of the model. This result allows reactivation to be done immediately, without an adverse effect on the smoothness of the solution.

After reactivation the strains and the deformation gradients are based on the displacements subsequent to the moment of reactivation, rather than on their total displacements. Thus, the current configuration at the start of the reactivation step is the new initial configuration for the element.

This kind of reactivation usually is used to model the creation of an undeformed and unstrained region of the model that is sharing a boundary with another, possibly stressed, deformed region. For example, in tunnel excavation an unstressed tunnel liner is added to line the walls of an already deformed tunnel (see *Stress-free element reactivation*).

Input File Usage: Use the following option to reactivate elements in a strain-free state:

**MODEL CHANGE*, ADD=STRAIN FREE (default)

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Small-Displacement Analysis

In small-displacement analysis the displacements at reactivation are considered to be small; therefore, volume, mass, initial length, and orientation directions do not change.

Large-Displacement Analysis

In large-displacement analysis the new configuration can be significantly different from the original configuration specified in the model definition. The change in configuration may result from large deformation or rigid body motion. For the nodes of the reactivated elements to be in the correct position upon reactivation, these nodes must be shared by elements that are not removed. Otherwise, the nodes of the removed elements remain at the location occupied at the time of removal. For cases where an enclosed region of material is reactivated, the shared-node restriction may require that a duplicate set of elements whose material properties do not influence the stress solution be defined on top of the removed elements. These duplicate elements provide a means of tracking the position of the nodes of the removed elements.

Upon reactivation an element can have a significantly different volume or mass, so the mass matrix is reformed for the element. Any local orientations applicable to the element are redefined on the new configuration. For shell and membrane elements, however, the thickness of the reactivated elements is the thickness as specified at the start of the analysis by the element's section definition, a nodal thickness definition (*Nodal Thicknesses*), or an import definition (*About Transferring Results between Abaqus Analyses*).

The current normals on structural elements at the moment of reactivation become new initial normals for that element. The current normal is the element's original normal (as specified in the model definition) rotated by the nodal rotation at the moment of reactivation. This scheme preserves the angle between the normals of reactivated elements and those of the elements with which they share nodes. (Usually, this angle should be zero and the normals should be identical, such as when a strain-free layer is added to an already deformed shell or beam. This can be achieved by ensuring that the normals are identical in the model definition.) If the reactivated structural elements share nodes with only non-structural elements (elements that do not provide stiffness to rotational degrees of freedom), duplicate structural elements are required so that the rotational degrees of freedom at the shared nodes will follow the deformation and rigid body motion before reactivation.

In a large-displacement analysis an element that is being reactivated strain free fits into whatever configuration is given by its nodes at the moment of reactivation. You must ensure that this configuration is meaningful and is not severely distorted. Abaqus/Standard will apply geometry checks on the reactivated elements; these checks are the same as the checks that are done in the analysis input file processor. Warnings are printed in the message file if the elements seem inappropriately distorted; and error messages are given if the distortion is severe, in which case the analysis will be stopped. If a geometry check on an element produces a warning or an error message, its current coordinates—and normals if applicable—are printed to the message file for your inspection. The current coordinates can be printed for all elements being reactivated by requesting detailed printout for element removal/reactivation, as explained in *The Abaqus/Standard Message File*.

Reactivating Axisymmetric Elements

Abaqus/Standard will not stop the analysis if an axisymmetric element has a very small negative radial coordinate at reactivation (if the magnitude of the radial coordinate is less than 10^{-4} times the average element length). In this case a warning is printed, and a radial coordinate of zero is assumed. If the radial coordinate is negative and larger than 10^{-4} times the average element length in magnitude, the analysis will stop.

For axisymmetric-asymmetric elements (SAXA and CAXA) the displacements at reactivation are considered small even in large-displacement analysis because these elements require an axisymmetric original configuration, but the configuration given by the nodes of these elements at reactivation would not, in general, be axisymmetric. Therefore, the original configuration is assumed not to change for these elements.

Reactivating Coupled Temperature-Displacement and Coupled Thermal-Electrical-Structural Elements

In a fully coupled temperature-displacement analysis and a fully coupled thermal-electrical-structural analysis, continuum elements attain their full mechanical stiffness immediately upon strain-free reactivation; however, to ensure smoothness of the solution, thermal conductivity is ramped up from zero over the step.

Reactivating Spring Elements and Substructures

If spring elements or substructures are reactivated “without strain,” the configuration at the moment of reactivation represents the zero-displacement state of the element; the forces in the spring or substructures are based on relative displacements subsequent to the moment of reactivation.

Reactivation with Strain

Elements reactivated with strain start in an annealed state unless reactivation in a nonvirgin state is specified, as described below.

The following scheme is implemented for the elements during the reactivation step: Let u^g represent the displacements of the nodes of this element, which are the displacements as shared by the rest of the model or as specified by boundary conditions. In general, these displacements can vary with time over the reactivation step. At any time in the reactivation step Abaqus/Standard enforces displacements, u^e, for the element:

$$u^e = \alpha(t)\, u^g,$$

where $\alpha(t)$ is a parameter that ramps linearly from 0 to 1 during the step. Thus, during the step the displacements felt by the reactivated elements ramp up to their actual values. To produce a consistent stiffness matrix, the element stiffness is also multiplied by $\alpha(t)$; therefore, the rest of the model experiences the reactivated elements as though their stiffnesses were ramped up during the step.

This ramping up of displacements instead of direct ramping up of element forces ensures that the strain in the element ramps up from zero to the strain given by the displacement of its nodes. This gradual ramping up of strains is desirable so that the response of history-dependent materials can be integrated gradually.

Subsequent to the end of the reactivation step, the strains in reactivated elements correspond to the displacements of their nodes from their initial configuration, rather than to their displacements since the moment of reactivation. This is appropriate, for example, in the refueling of a nuclear reactor, where the new fuel assembly must conform to the distortion of its old neighbors.

This reactivation scheme does not work for the rotations of shell elements that have five degrees of freedom per node because a total rotation is not stored at those nodes. Consequently, reactivation with strain is not allowed for these elements.

If an element is reactivated with strain after having been previously reactivated strain free, the strains are based on the displacements from the configuration in which the element was reactivated strain free (because this defined the new initial configuration for the element). In this case the u^g in the formula above should be interpreted as the displacement of the node relative to the position in which the element was reactivated strain free.

Input File Usage: Use the following option to reactivate elements with strain:

**MODEL CHANGE*, ADD=WITH STRAIN

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**; toggle on **Reactivated elements with strain (when applicable)**

Reactivating Elements with Rebar

Rebars are reactivated strain free or with strain exactly like the element in which they are defined. The annealing that takes place upon reactivation is also applied to rebars in the model. Reactivation of rebars can also be done in a nonvirgin state.

Reactivating Other Element Types

During reactivation of all element types other than stress/displacement elements, substructures, and contact elements, the nodal forces caused by stress in the element and by distributed loads are scaled by a value that ramps from zero to one during the reactivation step. (The nodal fluxes are scaled similarly for heat transfer elements.) In effect this scaling ramps the element stiffness up from zero during the step; for elements with mass or damping this scaling also ramps up the mass or damping during the step.

During the reactivation step the thermal conductivity of heat transfer elements and the permeability of pore pressure elements are ramped up from zero over the step.

User-defined elements can be removed and reactivated. User subroutine *UEL* is not called in steps in which the element is being removed or has already been removed.

Input File Usage: **MODEL CHANGE*, ADD

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Removing and Reactivating Contact Pairs

You can remove specified secondary and main surfaces from the model in a general analysis step. Contact pair removal and reactivation is explained in *Removing and Reactivating Contact Pairs*.

Input File Usage: **MODEL CHANGE*, TYPE=CONTACT PAIR, REMOVE or ADD

Abaqus/CAE Usage: Use the following option to remove contact pairs:

Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)** or **Self-contact (Standard)**: toggle off **Active in this step**

Use the following option to reactivate contact pairs:

Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)** or **Self-contact (Standard)**: toggle on **Active in this step**

Removing and Reactivating Contact Elements

Contact elements are removed and reactivated by Abaqus/Standard in the same way as contact pairs, as described in *Removing and Reactivating Contact Pairs*.

Input File Usage: **MODEL CHANGE*, TYPE=ELEMENT, REMOVE or ADD

Abaqus/CAE Usage: Use the following option to remove contact elements:

Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Deactivated in this step**

Use the following option to reactivate contact elements:

Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Modeling Issues

In some cases element removal/reactivation may cause numerical problems. The following guidelines can be used to reduce the chance of difficulty:

- If elements are removed in a static stress analysis and this removal leaves a region of the model with an unconstrained rigid body mode, solver problems will occur and the analysis most likely will fail to converge. Therefore, ensure that the remainder of the model is constrained sufficiently.

- If elements that are connected to a contact pair are removed, the contact pair should also be removed to avoid solver problems.
- If all elements attached to a node constrained with a multi-point constraint or a linear constraint equation are being removed, this node should be the dependent node of the multi-point constraint or linear constraint equation.

In some cases element removal may cause Abaqus/Standard to report extra unconnected regions in the message file. These messages can be safely ignored.

Removing or Reactivating Elements and Contact Pairs in a Restart Analysis

Elements or contact pairs can be removed or reactivated in a restart analysis (*Restarting an Analysis*) only if elements or contact pairs were removed or reactivated in the original analysis. In situations where it is expected that the addition or removal of elements or contact pairs will be required in a restart analysis, but there is no such need in the original analysis, you must activate element or contact pair removal/reactivation in the original analysis. Activating this capability does not add or remove any elements or contact pairs; it only prepares Abaqus/Standard to allow for these changes in a subsequent restart analysis.

Input File Usage: Use the following option to activate element or contact pair removal/reactivation:

**MODEL CHANGE*, ACTIVATE

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Restart**

Procedures

Elements or contact pairs cannot be removed or reactivated in a linear perturbation step (see *General and Perturbation Procedures*) or in a static Riks step (see *Unstable Collapse and Postbuckling Analysis*). For elements to be absent in such steps, they must have been inactive at the end of the previous general analysis (nonperturbation) step.

Initial Conditions

When elements are added back into the model, they are usually assumed to be “annealed”; that is, they have zero plastic strain, creep strain, etc. and zero stress at the start of the step in which they are reactivated. It is possible to reactivate an element so that it starts with a nonzero stress, equivalent plastic strain, and, if relevant, backstress (in a nonvirgin state).

Reactivation in a Nonvirgin State

To reactivate elements with nonzero stress, define initial stress conditions (see *Initial Conditions*) to specify the required stress in the model definition. Then the elements must be removed in the first step of the analysis. When reactivated, they will have the initial stress specified. The reactivation is done immediately, so the initial stress (which is applied in full during the first increment) must be self-equilibrating to avoid convergence issues.

If the elements were not removed in the first step, if they were removed again after the first step, or if initial conditions were not specified for them, they will have zero stresses when reactivated.

In a similar manner a material can be reactivated with a nonzero initial equivalent plastic strain and, if relevant, backstress.

When elements are reactivated, any applied initial stress is not displayed in the zero increment frame.

Input File Usage: Use the following option to specify initial stress conditions:

**INITIAL CONDITIONS*, TYPE=STRESS

Use the following option to specify initial equivalent plastic strain and backstress:

**INITIAL CONDITIONS*, TYPE=HARDENING

Abaqus/CAE Usage: Use the following options to specify the initial stress conditions:

Load module: **Create Predefined Field**: **Step**: **Initial**, choose **Mechanical** for the **Category** and **Stress** for the **Types for Selected Step**

Use the following options to specify the initial equivalent plastic strain and backstress:

Load module: **Create Predefined Field**: **Step**: **Initial**, choose **Mechanical** for the **Category** and **Hardening** for the **Types for Selected Step**

Boundary Conditions

The nodal variables of removed elements are not changed when the elements are removed. You can reset these variables by defining a boundary condition while the elements are inactive (see *Boundary Conditions*).

Loads

Distributed and concentrated loads that are applied in an area where elements are removed or reactivated may need to be modified.

Distributed Loads

Any distributed loads, fluxes, flows, and foundations specified for inactive elements are also inactive. However, unless you explicitly remove them, records of these loads are still kept and are listed in the data (`.dat`) file as though the elements were still present. Continuation of loads across steps is not affected by removal; on element reactivation unremoved distributed loads are also reactivated.

By default, if a distributed load is applied to an element that is being reactivated in a step, the distributed load magnitude is scaled up linearly from zero to its end-of-step value during the step. If such a load is applied with an amplitude reference, the magnitude value given by the amplitude reference is scaled again by a value that ramps from zero to one throughout the step. This scheme ensures that reactivation has a smooth effect on the solution, even in cases where a distributed load with an amplitude reference on a reactivated element is carried over from a previous step.

Concentrated Loads

Concentrated loads or fluxes are not removed when the surrounding elements are removed; therefore, you must ensure that any concentrated loads or fluxes that are carried solely by removed elements are also removed. Otherwise, a solver problem will occur during the removal step (a force is applied to a degree of freedom with zero stiffness). Concentrated loads or fluxes should be ramped up when they are reintroduced along with reactivated elements.

Predefined Fields

The nodal variables of removed elements are not changed directly when the elements are removed. You can reset these variables by defining temperature or other predefined field variables while the elements are inactive (see *Predefined Fields*). For example, elements that are removed in a stress/displacement analysis can be

reintroduced at a different temperature by setting the temperatures at the nodes on these elements to the desired value while the elements are inactive due to removal.

Temperatures

The temperatures at the start of the reactivation step become the initial temperatures for reactivated elements; thermal strains (and, thus, also the thermal stresses) are based on the temperature change subsequent to the instant of reactivation (see *Thermal Expansion*).

Material Options

On annealing, compaction-related quantities—such as the yield stress in hydrostatic compression, p_c, in crushable foam plasticity (*Crushable Foam Plasticity Models*); the yield stress in hydrostatic compression, p_b, in cap plasticity (*Modified Drucker-Prager/Cap Model*); and the void volume fraction, f, in porous metal plasticity (*Porous Metal Plasticity*)—are reset to the values they had at the start of the analysis.

For porous materials the porosity, n, is reset to its initial value and the saturation, s, retains its value from the instant of removal (see *About Pore Fluid Flow Properties*).

Elements with a user-defined material type can be removed and reactivated; user subroutines *UMAT* and *UMATHT* are not called while the elements are inactive. On reactivation the stresses and strains in user subroutine *UMAT* are set to zero, and conductivity and heat fluxes defined in user subroutine *UMATHT* are ramped up from zero during the reactivation step. Solution-dependent state variables must be reset in user subroutine *UMAT*, *UMATHT*, or *SDVINI*, which will be called on reactivation.

Elements

Removal is not currently supported for rigid, cohesive, gasket, and piezoelectric elements. All other element types in Abaqus/Standard can be removed and reactivated. See *Choosing the Appropriate Element for an Analysis Type*.

Output

Output is not available for elements or contact surfaces that have been removed. Inactive elements and contact surfaces are visible in Abaqus/CAE.

Input File Template

```
*HEADING
…
*STEP
*STATIC
…
** Remove all elements in element set SIDE
*MODEL CHANGE, REMOVE
 SIDE,
** Remove contact pair (SECONDARY1, MAIN1)
*MODEL CHANGE, TYPE=CONTACT PAIR, REMOVE
 SECONDARY1, MAIN1
…
*END STEP
**
*STEP
*STATIC
…
** Reactivate elements in element set SIDE
```

```
*MODEL CHANGE, ADD=STRAIN FREE
 SIDE,
** Reactivate contact pair (SECONDARY1, MAIN1)
*MODEL CHANGE, TYPE=CONTACT PAIR, ADD
 SECONDARY1, MAIN1
…
*END STEP
```

Progressive Element Activation

Products: Abaqus/Standard

References:

- **ACTIVATE ELEMENTS*
- **ELEMENT PROGRESSIVE ACTIVATION*
- *UEPACTIVATIONVOL*
- *Accessing Abaqus Table Collections*
- *Using Abaqus/Standard Output Variable Identifiers*

Overview



Warning: Development of this capability and user interface is expected to evolve in subsequent releases. Therefore, models may not be upward compatible without modification.

Progressive element activation:

- can be used to activate elements during an analysis;
- can be used to apply eigenstrains during element activation;
- can be used only with solid and shell elements; and
- can be used only in heat transfer or static analyses.

Progressive Element Activation

You can activate elements in each increment of a step. You must first define the elements that can be activated during an analysis and then refer to them in each analysis step in which they can be activated. Elements for which the activation feature is turned on in a step can be activated by assigning a volume fraction of material to an element at the beginning of each increment.

Both full and partial element activation are supported. For full activation the material volume fraction added must be equal to 0 or 1 (that is, the status of an element can change only from inactive to fully active). For partial activation the material volume fraction added can be arbitrary; however, in practice the volume fraction in an element should not be too small to prevent numerical singularity problems.

In stress-displacement analyses it is assumed that the material added to an element is stress free. Therefore, for full activation the configuration at which an element is activated is the stress-free configuration from which the strains used to compute the material response are measured. For partial activation the newly added material and the material already present are at different states. To obtain the material response, Abaqus/Standard uses the rule of mixtures to compute homogenized state variables.

Specifying Elements for Activation

You must first define the elements that can be used for activation in the model in the same way that you define regular elements. Then you must assign the elements to a specific progressive element activation feature.

Input File Usage: Use the following option to define elements that can be activated during an analysis:

**ELEMENT PROGRESSIVE ACTIVATION*, NAME=*activation_name*, ELSET=*element_set_name*

Switching Off/On Progressive Element Activation in a Step

Elements that are assigned to a specific progressive element activation can be activated only in steps in which the feature is switched on.

Input File Usage: Use the following option to switch on progressive element activation in a step and optionally specify the table collection name:

**ACTIVATE ELEMENTS*, ACTIVATION=*activation_name*

By default, progressive element activation is switched off in the step. You can repeat the option in each step as many times as necessary to turn on multiple activations.

Activating Elements

To activate elements within a step, you must assign the volume fraction of the material to the element in user subroutine *UEPACTIVATIONVOL*, which is called at the beginning of each increment. If a table collection has been specified for this activation, the data from parameter tables can be accessed from the user subroutine (see *Accessing Abaqus Table Collections*).

Controlling the Behavior of Inactive Elements

By default, elements that are inactive do not contribute to the overall response of the model and their degrees of freedom are not part of the solution (except for degrees of freedom at nodes shared with active elements). In stress-displacement analyses this approach works only if displacements are relatively small. If this is not the case, the inactive elements may become excessively distorted before they are activated, which may cause convergence difficulties or produce poor results. In this case you can allow the inactive elements to follow the deformation, which prevents excessive element deformation.

Input File Usage: Use the following option to specify that inactive elements should follow the deformation:

**ELEMENT PROGRESSIVE ACTIVATION*, FOLLOW DEFORMATION=YES

Use the following option to specify that inactive elements should not move:

**ELEMENT PROGRESSIVE ACTIVATION*, FOLLOW DEFORMATION=NO

Scaling the Material Properties of the Inactive Elements

When you specify that inactive elements should follow the deformation, all the elements in the model contribute to the response. However, you can scale the material properties of the inactive elements by specifying a preactivation coefficient. If the value of the scaling coefficient is sufficiently small, the contribution from the inactive elements does not markedly affect the solution and at the same time the elements follow the deformation and do not deform excessively. The default value of the preactivation coefficient is 10^{-4}.

Input File Usage: Use the following option to specify the preactivation coefficient:

**SECTION CONTROLS*, PREACTIVATION SCALING=*preactivation_coefficient*

Applying Initial Thermal Strains

When an element is activated, the initial thermal strains, ϵ_0^{th}, are computed with respect to the initial temperature. This might result in large values of strains applied instantaneously, which is equivalent to applying instantaneous loads. Such loads might cause convergence problems that will not be resolved by reducing the time increment. Abaqus avoids these convergence problems by specifying that thermal strains are ramped up by default instead of being applied instantaneously.

The initial thermal strains are ramped up over time according to the formula

$$\epsilon^{th} = \begin{cases} \frac{t-t_{act}}{\tau_{th}} \epsilon_0^{th}, & t_{act} \le t \le t_{act} + \tau_{th} \\ \epsilon_0^{th}, & t > t_{act} + \tau_{th} \end{cases}$$

where ϵ^{th} is the thermal strain applied, ϵ_0^{th} is the value of the thermal strain at the end of the increment at which the element is activated, t_{act} is the activation time, and τ_{th} is a user-specified expansion time constant. The default value of τ_{th} is 2 times the initial time increment. Specifying τ_{th}=0 causes the thermal strains to be applied instantaneously.

Input File Usage: Use the following option to specify the expansion time constant:

**ACTIVATE ELEMENTS*, EXPANSION TIME CONSTANT=τ_{th}

Applying Eigenstrains

Eigenstrain is a generic name given to inelastic strains such as thermal strain, plastic strain, phase transformation strain, and others. These strains can develop during various manufacturing processes such as welding, thermo-mechanical treatments, or additive manufacturing due to mechanical and thermal loads to which a material is subjected. If the distribution of these eigenstrains is known, it can be used to estimate the distortion and residual stresses in the body. In Abaqus, eigenstrains can be prescribed in user subroutine *UEPACTIVATIONVOL* to the new material that is added to an element. In addition, for solid elements the local orientation can be updated when the element is first activated.

As in the case of thermal strains, a sudden application of eigenstrains could lead to convergence problems. Therefore, Abaqus allows the eigenstrains to be ramped up linearly according to the formula:

$$\epsilon^{eig} = \begin{cases} \frac{t-t_{act}}{\tau_{eig}} \epsilon_0^{eig}, & t_{act} \le t \le t_{act} + \tau_{eig} \\ \epsilon_0^{eig}, & t > t_{act} + \tau_{eig} \end{cases},$$

where ϵ^{eig} is the eigenstrain applied, ϵ_0^{eig} is the value of the eigenstrain at the beginning of the increment at which the element is activated, t_{act} is the activation time, and τ_{eig} is a user-specified time constant. The default value of τ_{eig} is zero.

Input File Usage: Use the following option to specify the time constant:

**ACTIVATE ELEMENTS*, EIGEN TIME CONSTANT=τ_{eig}

Initial Configuration

In a static analysis the position of the nodes that are shared by active and inactive elements in general will change before the elements are activated. In this case the configuration at the time of element activation is different from

the original element configuration. This new configuration is assumed to be stress free, and the deformation from this configuration determines the stress in the element.

Procedures

Progressive element activation is supported only with the uncoupled heat transfer (see *Uncoupled Heat Transfer Analysis*) and static (see *Static Stress Analysis*) procedures.

Initial Conditions

Abaqus allows the initial volume fraction of material in an element to be specified. In addition, the initial temperatures are handled in a special way when progressive element activation is used in an analysis.

Defining Initial Volume Fraction

You can specify the initial values of volume fraction for elements that can be progressively activated. The volume fraction must be equal to 0 or 1, which means that the element can be either inactive or fully active at the beginning of an analysis.

Input File Usage: **INITIAL CONDITIONS*, TYPE=ACTIVATION

Initial Temperatures

In general, when elements are activated their state is set to the state at the beginning of the analysis. However, special handling is required for temperatures in a heat transfer analysis. In this case the temperatures at the integration points are interpolated from the nodal temperatures. Since inactive and active elements might share nodes, the nodal temperatures and, consequently, the temperatures at the integration points might be different from the specified initial values of the temperatures. In this case Abaqus generates a body heat flux at an integration point to compensate for this difference.

Boundary Conditions

If you specify that inactive elements should follow the deformation, the boundary conditions are applied to inactive nodes (since the degrees of freedom at these nodes are part of the solution). Otherwise, the boundary conditions are not applied to the inactive nodes until the elements to which they belong are activated.

Loads

Loads are not applied when an element is inactive; however, they are applied as soon as an element is activated. The load magnitude at activation has the value corresponding to the time at activation, which means that the load magnitude can increase suddenly, what might cause convergence problems.

Elements

Progressive element activation is supported only for solid continuum elements and shell elements. However, for shell elements only full activation is supported. If a volume fraction of material smaller than one is assigned to a shell element, Abaqus automatically changes the value to one.

Output

In addition to the standard output identifiers available in Abaqus/Standard (*Using Abaqus/Standard Output Variable Identifiers*), the following variable has special meaning when progressive element activation is specified:

EACTIVE	Volume fraction of the material in the current element.
EEIG	All components of the eigenstrain.
UACT	All physical displacement components, including rotations at nodes with rotational degrees of freedom, measured from the time the node is activated.
URACT	All rotational displacement components measured from the time the node is activated.
UTACT	All translational displacement components measured from the time the node is activated.

Introducing a Geometric Imperfection into a Model

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Unstable Collapse and Postbuckling Analysis*
- **IMPERFECTION*

Overview

A geometric imperfection pattern:

- is generally introduced in a model for a postbuckling load-displacement analysis;
- can be defined as a linear superposition of buckling eigenmodes obtained from a previous eigenvalue buckling prediction or eigenfrequency extraction analysis performed with Abaqus/Standard;
- can be based on the solution obtained from a previous static analysis performed with Abaqus/Standard; or
- can be specified directly.

General Postbuckling Analysis

In Abaqus/Standard the Riks method (*Unstable Collapse and Postbuckling Analysis*) can be used to solve postbuckling problems, both with stable and unstable postbuckling behavior. However, the exact postbuckling problem often cannot be analyzed directly due to the discontinuous response (bifurcation) at the point of buckling. To analyze a postbuckling problem, you must turn it into a problem with continuous response instead of bifurcation, which can be accomplished by introducing a geometric imperfection pattern in the "perfect" geometry so that there is some response in the buckling mode before the critical load is reached.

Introducing Geometric Imperfections

Imperfections are usually introduced by perturbations in the geometry. Abaqus offers three ways to define an imperfection: as a linear superposition of buckling eigenmodes, from the displacements of a static analysis, or by specifying the node number and imperfection values directly. Only the translational degrees of freedom are modified. Abaqus will then calculate the normals using the usual algorithm based on the perturbed coordinates. Unless the precise shape of an imperfection is known, an imperfection consisting of multiple superimposed buckling modes can be introduced (*Eigenvalue Buckling Prediction*).

The usual approach involves two analysis runs with the same model definition, using Abaqus/Standard to establish the probable collapse modes and either Abaqus/Standard or Abaqus/Explicit to perform the postbuckling analysis:

1. In the first analysis run perform an eigenvalue buckling analysis with Abaqus/Standard on the "perfect" structure to establish probable collapse modes and to verify that the mesh discretizes those modes accurately. Write the eigenmodes in the default global system to the results or output database file as nodal data.
2. In the second analysis run use Abaqus/Standard or Abaqus/Explicit to introduce an imperfection in the geometry by adding these buckling modes to the "perfect" geometry. The lowest buckling modes are frequently assumed to provide the most critical imperfections, so usually these are scaled and added to the perfect geometry to create the perturbed mesh. The imperfection thus has the form

$$\Delta x_i = \sum_{i=1}^{M} w_i \phi_i,$$

where ϕ_i is the i^{th} mode shape and w_i is the associated scale factor.

You must choose the scale factors of the various modes; usually (if the structure is not imperfection sensitive) the lowest buckling mode should have the largest factor. The magnitudes of the perturbations used are typically a few percent of a relative structural dimension such as a beam cross-section or shell thickness.

3. Use either Abaqus/Standard or Abaqus/Explicit to perform the postbuckling analysis.
   - In Abaqus/Standard perform a geometrically nonlinear load-displacement analysis of the structure containing the imperfection using the Riks method. In this way the Riks method can be used to perform postbuckling analyses of "stiff" structures that show linear behavior prior to buckling, if perfect. By performing a load-displacement analysis, other important nonlinear effects, such as material inelasticity or contact, can be included.
   - In Abaqus/Explicit perform a postbuckling analysis on the perturbed structure.

Abaqus imports imperfection data through the user node labels. Abaqus does not check model compatibility between both analysis runs. Node set definitions in the original model and the model with the imperfection may be different. Care must be taken for models in which Abaqus generates additional nodes (for example, the nodes generated for contact surfaces on 20-node brick elements). In such cases you have to ensure that the models for both analysis runs are identical and that the nodal information for the generated nodes is written to the results file.

If the model is defined in terms of an assembly of part instances, the part (`.prt`) file from the original analysis is required to read the eigenmodes from the results file. Both the original model and the subsequent model must be defined consistently in terms of an assembly of part instances.

Defining an Imperfection Based on Eigenmode Data

To define an imperfection based on the superposition of weighted mode shapes, specify the results (`.fil`) file or an output database (`.odb` or `.sim`) file and step from a previous eigenfrequency extraction or eigenvalue buckling prediction analysis. The file extension is optional. However, if both a results file and output database file exist, the results file is used. If there is no results file but both the `.odb` and `.sim` files exist, the `.odb` file is used. Optionally, you can import eigenmode data for a specified node set.

Input File Usage: **IMPERFECTION*, FILE=*file*, STEP=*step*, NSET=*name*

Defining an Imperfection Based on Static Analysis Data

To define an imperfection based on the deformed geometry of a previous static analysis (*Unstable Collapse and Postbuckling Analysis*), specify the results (`.fil`) file or an output database (`.odb` or `.sim`) file and step (and, optionally, the increment number) from a previous static analysis. (If the increment number is not specified, Abaqus will read data from the last increment available for the specified step in the results or output database file.) The file extension is optional. However, if both a results file and output database file exist, the results file is used. If there is no results file but both the `.odb` and `.sim` files exist, the `.odb` file is used. Optionally, you can import modal data for a specified node set.

Input File Usage: **IMPERFECTION*, FILE=*file*, STEP=*step*, INC=*inc*, NSET=*name*

Defining an Imperfection Directly

You can specify the imperfection directly as a table of node numbers and coordinate perturbations in the global coordinate system or, optionally, in a cylindrical or spherical coordinate system. Alternatively, you can read the imperfection data from a separate input file.

Input File Usage: **IMPERFECTION*, SYSTEM=*name*, INPUT=*input file*

If no input file is specified, Abaqus assumes that the data follow the option.

Imperfection Sensitivity

The response of some structures depends strongly on the imperfections in the original geometry, particularly if the buckling modes interact after buckling occurs. Hence, imperfections based on a single buckling mode tend to yield nonconservative results. By adjusting the magnitude of the scaling factors of the various buckling modes, the imperfection sensitivity of the structure can be assessed. Normally, a number of analyses should be conducted to investigate the sensitivity of a structure to imperfections. Structures with many closely spaced eigenmodes tend to be imperfection sensitive, and imperfections with shapes corresponding to the eigenmode for the lowest eigenvalue may not give the worst case.

The imperfect structure will be easier to analyze if the imperfection is large. If the imperfection is small, the deformation will be quite small (relative to the imperfection) below the critical load. The response will grow quickly near the critical load, introducing a rapid change in behavior.

On the other hand, if the imperfection is large, the postbuckling response will grow steadily before the critical load is reached. In this case the transition into postbuckled behavior will be smooth and relatively easy to analyze.

Input File Template

The following example illustrates a postbuckling analysis of a structure with an imperfection defined by a linear superposition of the buckling eigenmodes and involves two analysis runs with the same model definition.

The initial analysis run performs an eigenvalue buckling analysis with Abaqus/Standard to establish the probable collapse modes and writes them to the results file.

```
*HEADING
Initial analysis run to write the buckling modes to the results file
*NODE
Data lines to define initial “perfect” geometry
...
**
*STEP
*BUCKLE
Data lines to define the number of buckling eigenmodes
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
Data lines to specify the reference load, P_ref
*NODE FILE, GLOBAL=YES, LAST MODE=n
U
*END STEP
```

The second analysis run introduces the imperfection and performs a postbuckling analysis employing the modified Riks method in Abaqus/Standard.

```
*HEADING
Second analysis run to define the imperfection and perform the postbuckling analysis
*NODE
```

```
Data lines to define initial “perfect” geometry
…
*IMPERFECTION, FILE=results_file, STEP=step
Data lines specifying the mode number and its associated scale factor
…
**
*STEP, NLGEOM
*STATIC, RIKS
Data line to define incrementation and stopping criteria
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
Data lines to specify reference loading, P_ref
*END STEP
```

An alternative second analysis run introduces the imperfection and performs a postbuckling analysis with Abaqus/Explicit.

```
*HEADING
Second analysis run to define the imperfection and perform the postbuckling analysis
*NODE
Data lines to define initial “perfect” geometry
…
*IMPERFECTION, FILE=results_file, STEP=step
Data lines specifying the mode number and its associated scale factor
…
**
*STEP
*DYNAMIC, EXPLICIT
Data line to define the time period of the step.
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
*END STEP
```

Fracture Mechanics

In this section:

- *About Fracture Mechanics*
- *Contour Integral Evaluation*
- *Crack Propagation Analysis*

About Fracture Mechanics

Overview

Abaqus/Standard provides several methods for performing fracture mechanics studies.

The following methods are available:

Onset of cracking

The onset of cracking can be studied in quasi-static problems by using contour integrals (*Contour Integral Evaluation*). The J-integral, the C_t-integral (for creep), the stress intensity factors for both homogeneous materials and interfacial cracks, the crack propagation direction, and the T-stress are calculated by Abaqus/Standard. Contour integrals can be used in two- or three-dimensional problems. In these types of problems, focused meshes are generally required and the propagation of a crack is not studied.

Crack propagation

The crack propagation capability allows quasi-static, including low-cycle fatigue, crack growth along predefined paths to be studied (*Crack Propagation Analysis*). Cracks debond along user-defined surfaces. Several crack propagation criteria are available, and multiple cracks can be included in the analysis. Contour integrals can be requested in crack propagation problems.

Line spring elements

Part-through cracks in shells can be modeled inexpensively by using line spring elements in a static procedure, as explained in *Line Spring Elements for Modeling Part-through Cracks in Shells*.

Extended finite element method (XFEM)

XFEM models a crack as an enriched feature by adding degrees of freedom in elements with special displacement functions (*Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*). XFEM does not require the mesh to match the geometry of the discontinuities. It can be used to simulate initiation and propagation of a discrete crack along an arbitrary, solution-dependent path without the requirement of remeshing. You can also use XFEM to perform contour integral evaluation for an arbitrary stationary surface crack without the need to define a conforming mesh around the crack tip.

Fatigue crack growth with adaptive remeshing

Fatigue crack growth with adaptive remeshing is based on Abaqus fracture/fatigue technology in conjunction with tetrahedral element–based adaptive remeshing technology (*Fatigue Crack Propagation with Adaptive Remeshing*). The stress intensity factors are calculated based on the conventional contour integral method (*Contour Integral Evaluation*). The crack growth is governed by Paris' law (*Fatigue Crack Growth Laws for Linear Elastic Materials*). The tetrahedral element–based adaptive remeshing technology is utilized to create a new mesh after each increment of crack growth. The new mesh conforms to the updated crack front. The overall fatigue workflow is semiautomated and involves automatic execution (with no user intervention) of a series of successive Abaqus fatigue analyses with progressively longer crack lengths.

References:

Contour Integral Evaluation

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Fracture Mechanics*
- **CONTOUR INTEGRAL*
- *Output to the Output Database*
- *Using contour integrals to model fracture mechanics*

Overview

Abaqus/Standard offers the evaluation of several parameters for fracture mechanics studies based on either the conventional finite element method or the extended finite element method (XFEM, see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*):

- the *J*-integral, which is widely accepted as a quasi-static fracture mechanics parameter for linear material response and, with limitations, for nonlinear material response;
- the C_t-integral, which has an equivalent role to the *J*-integral in the context of time-dependent creep behavior (*Rate-Dependent Plasticity: Creep and Swelling*) in a quasi-static step (*Quasi-Static Analysis*);
- the stress intensity factors, which are used in linear elastic fracture mechanics to measure the strength of the local crack-tip fields;
- the crack propagation direction—that is, the angle at which a preexisting crack propagates; and
- the *T*-stress, which represents a stress parallel to the crack faces and is used as an indicator of the extent to which parameters like the *J*-integral are useful characterizations of the deformation field around the crack.

Contour integrals:

- are output quantities—they do not affect the results;
- can be requested only in general analysis steps;
- can be used only with two-dimensional quadrilateral elements, three-dimensional brick elements, or three-dimensional second-order tetrahedral elements when used with the conventional finite element method;
- can be evaluated without requiring a detailed conforming mesh around the crack tips when used with XFEM; and
- are currently available only for first-order or second-order tetrahedral and first-order brick elements with isotropic elastic material when used with XFEM.

Contour Integral Evaluation

Abaqus/Standard offers two different ways to evaluate the contour integral. The first approach is based on the conventional finite element method, which typically requires you to conform the mesh to the cracked geometry, to explicitly define the crack front, and to specify the virtual crack extension direction. Detailed focused meshes are generally required, and obtaining accurate contour integral results for a crack in a three-dimensional curved surface can be quite cumbersome. The extended finite element method (XFEM) alleviates these shortcomings. XFEM does not require the mesh to match the cracked geometry. The presence of a crack is ensured by the special enriched functions in conjunction with additional degrees of freedom. You must, however, generate a sufficient number of elements around the crack front to obtain path-independent contours, particularly in the

region with high crack front curvature. This approach also removes the requirement for explicitly defining the crack front or specifying the virtual crack extension direction when evaluating the contour integral. The data required for the contour integral are determined automatically based on the level set signed distance functions at the nodes in an element (see *Modeling Discontinuities as an Enriched Feature Using the Extended Finite Element Method*).

Several contour integral evaluations are possible at each location along a crack. In a finite element model each evaluation can be thought of as the virtual motion of a block of material surrounding the crack tip (in two dimensions) or surrounding each node along the crack line (in three dimensions). Each block is defined by contours, where each contour is a ring of elements completely surrounding the crack tip or the nodes along the crack line from one crack face to the opposite crack face. These rings of elements are defined recursively to surround all previous contours.

Abaqus/Standard automatically finds the elements that form each ring from the regions defined as the crack tip or crack line. Each contour provides an evaluation of the contour integral. The possible number of evaluations is the number of such rings of elements for two-dimensional quadrilateral and three-dimensional brick elements. For tetrahedral elements, you must specify a small radius within which rings of elements are identified for fracture mechanics studies. A refined mesh is required to define the rings of elements around the crack front, especially in a region near the external free surfaces. In a case where the crack front intersects the external free surface in a model with tetrahedral elements at an angle not equal to 90°, you should specify surface normals at all the crack tip nodes that lie on the external free surfaces (see *Normal Definitions at Nodes*). This action ensures that the tangential directions of the crack front at those locations are estimated accurately for contour integral evaluation. The default value of the ring radius is twice the typical element characteristic length along the crack front, which works well for most problems. You must specify the number of contours to use in calculating contour integrals. In addition, you must specify the type of contour integral to calculate, as described below. By default, Abaqus/Standard calculates the *J*-integral.

You can assign a name to a crack that is used to identify the contour integral values in the data file and in the output database file. The name is also used by Abaqus/CAE to request contour integral output. If you are using the conventional finite element method and do not specify a crack name, by default Abaqus/Standard generates crack numbers that follow the order in which the cracks are defined. If you are using XFEM, you must set the crack name equal to the name assigned to the enriched feature. Both the domain integral method and the line integral method are supported when you evaluate the contour integral using XFEM.

Input File Usage:

Use the following option to evaluate the contour integral with the conventional finite element method for two-dimensional quadrilateral and three-dimensional brick elements:

**CONTOUR INTEGRAL*, CRACK NAME=*crack name*, CONTOURS=*n*, TYPE=*integral_type*

Use the following option to evaluate the contour integral with the conventional finite element method for three-dimensional tetrahedral elements:

**CONTOUR INTEGRAL*, CRACK NAME=*crack name*, CONTOURS=*n*, TYPE=*integral_type*, RING RADIUS=*r*

Use one of the following options to evaluate the contour integral with XFEM by using the domain integral method or the line integral method:

**CONTOUR INTEGRAL*, CRACK NAME=*crack name*, XFEM=DOMAIN, CONTOURS=*n*, TYPE=*integral_type*

**CONTOUR INTEGRAL*, CRACK NAME=*crack name*, XFEM=LINE (default), CONTOURS=*n*, TYPE=*integral_type*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **Name:** *crack name*, **Type: Contour integral** or **XFEM**
Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type:** *integral_type*

Domain Integral Method

Using the divergence theorem, the contour integral can be expanded into an area integral in two dimensions or a volume integral in three dimensions, over a finite domain surrounding the crack. This domain integral method is used to evaluate contour integrals in Abaqus/Standard. The method is quite robust in the sense that accurate contour integral estimates are usually obtained even with quite coarse meshes. The method is robust because the integral is taken over a domain of elements surrounding the crack and because errors in local solution parameters have less effect on the evaluated quantities such as J, C_t, the stress intensity factors, and the T-stress.

Requesting Multiple Contour Integrals

Contour integrals at several different crack tips in two dimensions or along several different crack lines in three dimensions can be evaluated at any time by repeating the contour integral request as often as needed in the step definition. When you are using the conventional finite element method, you must specify the crack front and the direction of virtual crack extension (or the normal to the crack plane if this normal is constant) for each crack tip or crack line, as described below. When you are using XFEM, you do not need to specify the crack front or the virtual crack extension direction because they will be determined by Abaqus/Standard. However, you must set each crack name equal to the corresponding enriched feature, with each enriched feature consisting of only one crack. In addition, regardless of whether you are using either the conventional finite element method or XFEM, you must specify the number of contours to be calculated for each integral.

J-Integral

The J-integral is usually used in rate-independent quasi-static fracture analysis to characterize the energy release associated with crack growth. It can be related to the stress intensity factor if the material response is linear.

The J-integral is defined in terms of the energy release rate associated with crack advance. For a virtual crack advance $\lambda(s)$ in the plane of a three-dimensional fracture, the energy release rate is given by

$$\overline{J} = \int_A \lambda(s)\,\mathbf{n}\cdot\mathbf{H}\cdot\mathbf{q}\,dA,$$

where dA is a surface element along a vanishing small tubular surface enclosing the crack tip or crack line, $\mathbf{n}$ is the outward normal to dA, and $\mathbf{q}$ is the local direction of virtual crack extension. $\mathbf{H}$ is given by

$$\mathbf{H} = \left(W\mathbf{I} - \boldsymbol{\sigma}\cdot\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right).$$

For elastic material behavior W is the elastic strain energy; for elastic-plastic or elasto-viscoplastic material behavior W is defined as the elastic strain energy density plus the plastic dissipation, thus representing the strain energy in an "equivalent elastic material." Therefore, the J-integral calculated is suitable only for monotonic loading of elastic-plastic materials.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=J

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: J-integral**

Domain Dependence

The *J*-integral should be independent of the domain used provided that the crack faces are parallel to each other, but *J*-integral estimates from different rings might vary because of the approximate nature of the finite element solution. Strong variation in these estimates, commonly called domain dependence or contour dependence, typically indicates an error in the contour integral definition. Gradual variation in these estimates might indicate that a finer mesh is needed or, if plasticity is included, that the contour integral domain does not completely include the plastic zone. If the "equivalent elastic material" is not a good representation of the elastic-plastic material, the contour integrals will be domain independent only if they completely include the plastic zone. Since it is not always possible to include the plastic zone in three dimensions, a finer mesh may be the only solution.

If the first contour integral is defined by specifying the nodes at the crack tip, the first few contours might be inaccurate. To check the accuracy of these contours, you can request more contours and determine the value of the contour integral that appears approximately constant from one contour to the next. The contour integral values that are not approximately equal to this constant should be discarded. In linear elastic problems the first and second contours typically should be ignored as inaccurate.

For some three-dimensional models with an open crack front, the *J*-integral estimates might be inaccurate from the node sets (or elements in the case with XFEM) at the crack front ends. The resolution difficulty is compounded by the skewness of the outmost layer of elements. This accuracy loss is confined only to the contour integrals at the front ends and has no effect on the accuracy of the contour integral values at the neighboring node sets (or elements in the case with XFEM) along the crack front.

Including the Effect of a Residual Stress Field on *J*-Integral Evaluation

A residual stress field often occurs in a structure; for example, as a result of service loads that produce plasticity, a metal forming process in the absence of an anneal treatment, thermal effects, or swelling effects. When the residual stresses are significant, the standard definition of the *J*-integral as described above might lead to a path-dependent value. To ensure its path independence, the *J*-integral evaluation must include an additional term that accounts for the residual stress field. In Abaqus/Standard the problem with a residual stress field is treated as an initial strain problem. If the total strain is written as the sum of mechanical strain, $\boldsymbol{\varepsilon}^m$, and initial strain, $\boldsymbol{\varepsilon}^o$; that is,

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^m + \boldsymbol{\varepsilon}^o,$$

a path-independent energy release rate in the presence of a residual stress field is given by

$$\overline{J} = \int_A \lambda(s)\,\mathbf{n}\cdot\left(W\mathbf{I} - \boldsymbol{\sigma}\cdot\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)\cdot\mathbf{q}\,dA + \int_V \boldsymbol{\sigma} : \frac{\partial \boldsymbol{\varepsilon}^o}{\partial \mathbf{x}}\cdot\mathbf{q}\,dV,$$

where V is the domain volume enclosing the crack tip or crack line, W is defined as the mechanical strain energy density only,

$$W = \int_0^{\boldsymbol{\varepsilon}^m} \boldsymbol{\sigma} : d\boldsymbol{\varepsilon}^m,$$

and $\boldsymbol{\varepsilon}^o$ remains constant during the entire deformation.

The residual stress field can be specified by reading the stress data from a previous analysis step or by defining an initial condition (see *Defining Initial Stresses*). You specify the step number from which the stress data in the last available increment of the specified step will be considered as residual stresses. If the step number is set equal to zero (default), the residual stress field is defined by the initial condition definition. When XFEM is used, the residual stress field can be defined only with an initial condition definition.

The approach used to account for the residual stress effects was adopted from *Lei, O'Dowd, and Webster (2000)*.

Input File Usage: **CONTOUR INTEGRAL*, RESIDUAL STRESS STEP=*n*, TYPE=J

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: J-integral**

Allowing Non-Proportional Stressing

Usually, the proportional stressing condition is assumed to be satisfied when the contour integrals are calculated. In other words, this equation prevails:

$$\boldsymbol{\sigma} = \frac{\partial W}{\partial \boldsymbol{\varepsilon}^m}.$$

Otherwise, you should modify the path-independent energy release rate as follows:

$$\overline{J} = \int_A \lambda(s)\,\mathbf{n}\cdot\left(W\mathbf{I} - \boldsymbol{\sigma}\cdot\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)\cdot \mathbf{q}\,dA + \int_V \left(\boldsymbol{\sigma} : \frac{\partial \boldsymbol{\varepsilon}}{\partial \mathbf{x}} - \frac{\partial W}{\partial \mathbf{x}}\right)\cdot \mathbf{q}\,dV.$$

This approach used to account for the nonproportional stressing effects was adopted from *Lei (2005)*.

Input File Usage: **CONTOUR INTEGRAL*, NON-PROPORTIONAL STRESSING, TYPE=J

Abaqus/CAE Usage: Allowing non-proportional stressing is not supported in Abaqus/CAE.

C_t-Integral

The C_t-integral is supported with the conventional finite element method; however, it is not supported with XFEM.

The C_t-integral can be used for time-dependent creep behavior, where it characterizes creep crack deformation under certain creep conditions, including transient crack growth. C_t is, for example, proportional to the rate of growth of the crack-tip/crack-line creep zone for a stationary crack under small-scale creep conditions. Under steady-state creep conditions, when creep dominates throughout the specimen, C_t becomes path independent and is known as C^*. C_t-integrals should be requested only in a quasi-static step.

The C_t-integral is obtained by replacing the displacements with velocities and the strain energy density with the strain energy rate density in the *J*-integral expansion. The strain energy rate density is defined as

$$\dot{W} \overset{\text{def}}{=} \int_0^{\dot{\boldsymbol{\varepsilon}}} \boldsymbol{\sigma} : d\dot{\boldsymbol{\varepsilon}}.$$

$\dot{W}$ is not uniquely defined if multiple deformation mechanisms contribute to the strain rate. However, the creep mechanism will dominate within a zone surrounding a crack tip or crack line, so elastic and plastic contributions to $\dot{W}$ are negligible. The size of that zone depends on the extent of creep relaxation: the zone is initially small but eventually encompasses the entire specimen when steady-state creep is reached. Abaqus/Standard considers only creep in the calculation of $\dot{W}$. Neglecting elastic and plastic strain rates, the strain energy density for the power law creep model with time hardening form in Abaqus/Standard is

$$\dot{W} = \frac{n}{n+1} q \dot{\bar{\varepsilon}},$$

where n is the power law exponent, q is the equivalent von Mises stress, and $\dot{\bar{\varepsilon}}$ is the equivalent uniaxial strain rate.

For the hyperbolic-sine law an analytical expression of $\dot{W}$ is not available. For this law $\dot{W}$ is obtained by numerical integration; a five-point Gauss quadrature scheme gives reasonable accuracy in the range of realistic creep strain rates.

The domain integral method is used for C_t-integrals as described above for *J*-integrals.

For user-defined creep laws the strain energy rate density must be defined in user subroutine *CREEP*.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=C

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Ct-integral**

Domain Dependence

Prior to steady state C_t-integral estimates will exhibit domain dependence, even if the finite element mesh is sufficiently refined, because of the assumption of creep dominance within the domain specified. These C_t estimates should be extrapolated to zero radius to obtain an improved C_t estimate corresponding to a contour shrunk onto the crack tip or crack line (see *Ct-integral evaluation*).

Stress Intensity Factors

The stress intensity factors K_I, K_{II}, and K_{III} are usually used in linear elastic fracture mechanics to characterize the local crack-tip/crack-line stress and displacement fields. They are related to the energy release rate (the *J*-integral) through

$$J = \frac{1}{8\pi}\mathbf{K}^T \cdot \mathbf{B}^{-1} \cdot \mathbf{K},$$

where $\mathbf{K} = \lfloor K_I, K_{II}, K_{III} \rfloor^T$ are the stress intensity factors and $\mathbf{B}$ is called the prelogarithmic energy factor matrix. For homogeneous, isotropic materials $\mathbf{B}$ is diagonal, and the above equation simplifies to

$$J = \frac{1}{\overline{E}}\left(K_I^2 + K_{II}^2\right) + \frac{1}{2G}K_{III}^2,$$

where $\overline{E} = E$ for plane stress and $\overline{E} = E/\left(1-\nu^2\right)$ for plane strain, axisymmetry, and three dimensions. For an interfacial crack between two dissimilar isotropic materials,

$$J = \frac{1-\beta^2}{E^*}\left(K_I^2 + K_{II}^2\right) + \frac{1}{2G^*}K_{III}^2,$$

where

$$\frac{1}{E^*} = \frac{1}{2}\left(\frac{1}{\overline{E}_1} + \frac{1}{\overline{E}_2}\right), \qquad \frac{1}{G^*} = \frac{1}{2}\left(\frac{1}{G_1} + \frac{1}{G_2}\right);$$

$$\beta = \frac{G_1(\kappa_2-1) - G_2(\kappa_1-1)}{G_1(\kappa_2+1) + G_2(\kappa_1+1)};$$

$\kappa = 3 - 4\nu$ for plane strain, axisymmetry, and three dimensions; and $\kappa = (3-\nu)/(1+\nu)$ for plane stress.

Unlike their analogs in a homogeneous material, K_I and K_{II} are no longer the pure Mode I and Mode II stress intensity factors for an interfacial crack. They are simply the real and imaginary parts of a complex stress intensity factor.

Although the energy release rate is calculated directly in Abaqus/Standard, it is usually not straightforward to compute stress intensity factors from a known *J*-integral for mixed-mode problems. Abaqus/Standard provides an interaction integral method to compute the stress intensity factors directly for a crack under mixed-mode loading. This capability is available for linear isotropic and anisotropic materials. In the case of interfacial cracks, this capacity is limited to interfaces between dissimilar isotropic materials; it does not extend to cracks between dissimilar anisotropic materials. The theory is described in detail in *Stress intensity factor extraction*.

In this case the *J*-integrals calculated from the stress intensity factors will also be output. These *J*-integral values may be slightly different from those estimated by requesting the *J*-integral directly, due to the different algorithms used for the calculations.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=K FACTORS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**

Domain Dependence

The stress intensity factors have the same domain dependence features as the *J*-integral.

Including the Effect of a Residual Stress Field on Stress Intensity Factor Evaluation

An additional term is included to account for the residual stress field when calculating the stress intensity factors, as described in *Including the Effect of a Residual Stress Field on J-Integral Evaluation*.

Input File Usage: **CONTOUR INTEGRAL*, RESIDUAL STRESS STEP=*n*, TYPE=K FACTORS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: Stress intensity factors**

Crack Propagation Direction

For homogeneous, isotropic elastic materials the direction of cracking initiation can be calculated using one of the following three criteria: the maximum tangential stress criterion, the maximum energy release rate criterion, or the $K_{II} = 0$ criterion. K_{III} is not taken into account in any of these criteria.

Maximum Tangential Stress Criterion

Using either the condition $\partial \sigma_{\theta\theta} / \partial \theta = 0$ or $\tau_{r\theta} = 0$ (where r and θ are polar coordinates centered at the crack tip in a plane orthogonal to the crack line), we can obtain

$$\hat{\theta} = \cos^{-1}\left(\frac{3K_{II}^2 + \sqrt{K_I^4 + 8K_I^2 K_{II}^2}}{K_I^2 + 9K_{II}^2}\right),$$

where the crack propagation angle $\hat{\theta}$ is measured with respect to the crack plane and $\hat{\theta} = 0$ represents the crack propagation in the "straight-ahead" direction. $\hat{\theta} < 0$ if $K_{II} > 0$ while $\hat{\theta} > 0$ if $K_{II} < 0$ The crack propagation angle is measured from $\mathbf{q}$ to $\mathbf{n}$; that is, it is measured about the direction $-\mathbf{t}$, or counterclockwise measured from $\mathbf{q}$ in *Figure 1*.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=MTS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: Maximum tangential stress**

Maximum Energy Release Rate Criterion

This criterion postulates that a crack initially propagates in the direction that maximizes the energy release rate.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=MERR

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: Maximum energy release rate**

$K_{II} = 0$ Criterion

This criterion assumes that a crack initially propagates in the direction that makes $K_{II} = 0$.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=KII0

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: K11=0**

T-Stress

The *T*-stress component represents a stress parallel to the crack faces at the crack tip. Its magnitude can alter not only the size and shape of the plastic zone but also the stress triaxiality ahead of the crack tip. It is, therefore, a useful indicator of whether measures of the strength of the crack-tip singularity (such as the *J*-integral or the stress intensity factors) are useful in characterizing a crack under a particular loading. In a linear elastic analysis the *T*-stress should be calculated using loads equal to the loads in the elastic-plastic analysis. See *T-stress extraction* for more information.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, TYPE=T-STRESS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: T-stress**

Domain Dependence

In general, the *T*-stress has larger domain dependence or contour dependence than the *J*-integral and the stress intensity factors. Numerical tests suggest that the estimates from the first two rings of elements abutting the crack tip or crack line generally do not provide accurate results. Sufficient contours extending from the crack tip or crack line should be chosen so that the *T*-stress can be determined to be independent of the number of contours, within engineering accuracy. Particularly for axisymmetric models, the closer the crack tip is to the symmetry axis, the more refined the mesh in the domain should be to achieve path independence of the contour integral.

Including the Effect of a Residual Stress Field on T-Stress Evaluation

An additional term is included to account for the residual stress field when calculating the *T*-stress, as described in *Including the Effect of a Residual Stress Field on J-Integral Evaluation*.

Input File Usage: **CONTOUR INTEGRAL*, RESIDUAL STRESS STEP=*n*, TYPE=T-STRESS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: T-stress**

Defining the Data Required for a Contour Integral with the Conventional Finite Element Method

To request contour integral output with the conventional finite element method, you must define the crack front and specify the virtual crack extension direction.

Defining the Crack Front

You must specify the crack front; that is, the region that defines the first contour. Abaqus/Standard uses this region and one layer of elements surrounding it to compute the first contour integral. An additional layer of elements is used to compute each subsequent contour.

The crack front can be equivalent to the crack tip in two dimensions or the crack line in three dimensions; or it can be a larger region surrounding the crack tip or crack line, in which case it must include the crack tip or crack line.

If blunted crack tips are modeled, the crack front should include all the nodes going from one crack face to the other that would collapse onto the crack tip if the radius of the blunted tip were reduced to zero. Otherwise, the contour integral value will depend on the path until the contour region reaches the parallel crack faces.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*
Specify the crack front node set name on the data line; the format depends on the method you use to specify the virtual crack extension direction.

For two-dimensional cases only one crack front node set (the crack front at the crack tip) must be specified. For three-dimensional cases you must repeat the data line to specify the crack front for each node (or cluster of focused nodes) along the crack line in order from one end of the crack to the other, including the midside nodes of second-order elements; it is not permissible to skip nodes along the crack line.

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front

Defining the Crack Tip or Crack Line

By default, Abaqus/Standard defines the crack tip as the first node specified for the crack front and the crack line as the sequence of first nodes specified for the crack front. The first node is the node with the smallest node number, unless the node set is generated as unsorted. Alternatively, you can specify the crack-tip node or crack-line nodes directly. This specification plays a critical role for a three-dimensional crack with a blunt crack tip.

Abaqus/CAE cannot determine the crack tip or crack line automatically based on the specified crack front. However, if you select a point to define the crack front in two dimensions, the same point defines the crack tip; likewise, if you select edges to define the crack front in three dimensions, the same edges define the crack line. For all other cases you must define the crack tip or crack line directly.

Input File Usage: Use the following option to specify the crack-tip nodes directly:

**CONTOUR INTEGRAL*, CONTOURS=*n*, CRACK TIP NODES
Specify the crack front node set name and the crack tip node number or node set name on the data line; the format depends on the method you use to specify the virtual crack extension direction.

Repeat the data line for three-dimensional cases.

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front, then select the crack tip (in two dimensions) or crack line (in three dimensions)

Defining a Closed-Loop Crack Line

Sometimes a crack line might form a closed loop (for example, when modeling a full penny-shaped crack without invoking symmetry conditions). In such cases the finite element mesh in the crack-tip region can be created with or without seams; that is, linear constraint equations (*Linear Constraint Equations*) or multi-point constraints (*General Multi-Point Constraints*) might or might not be used to tie two layers of nodes together.

If a crack line forms a closed loop, the starting node set of the crack front can be chosen arbitrarily and the other node sets defining the crack front must go around the crack front sequentially. The last node set defining the crack front must be the same as the first node set. If a closed loop is formed by creating coincident nodes that are then tied together by linear constraint equations and multi-point constraints, the node sets must be specified in order starting from one of the node sets involved in the constraint equation or multi-point constraint and terminating with the other node set.

Specifying the Virtual Crack Extension Direction

You must specify the direction of virtual crack extension at each crack tip in two dimensions or at each node along the crack line in three dimensions by specifying either the normal to the crack plane, $\mathbf{n}$, or the virtual crack extension direction, $\mathbf{q}$.

If the virtual crack extension direction is specified to point into the material (parallel to the crack faces), the *J*-integral values calculated will be positive. Negative *J*-integral values are obtained when the virtual crack extension direction is specified in the opposite direction.

Specifying the Normal to the Crack Plane

The virtual crack extension direction can be defined by specifying the normal, $\mathbf{n}$, to the crack plane. In this case Abaqus/Standard will calculate a virtual crack extension direction, $\mathbf{q}$, that is orthogonal to the crack front tangent, $\mathbf{t}$, and the normal, $\mathbf{n}$. As shown in *Figure 1*, $\mathbf{q} = \mathbf{t} \times \mathbf{n}$ for a three-dimensional crack; for a two-dimensional crack, we simply have $q_x = -n_y$ and $q_y = n_x$. Specifying the normal implies that the crack plane is flat since only one value of $\mathbf{n}$ can be given per contour integral.

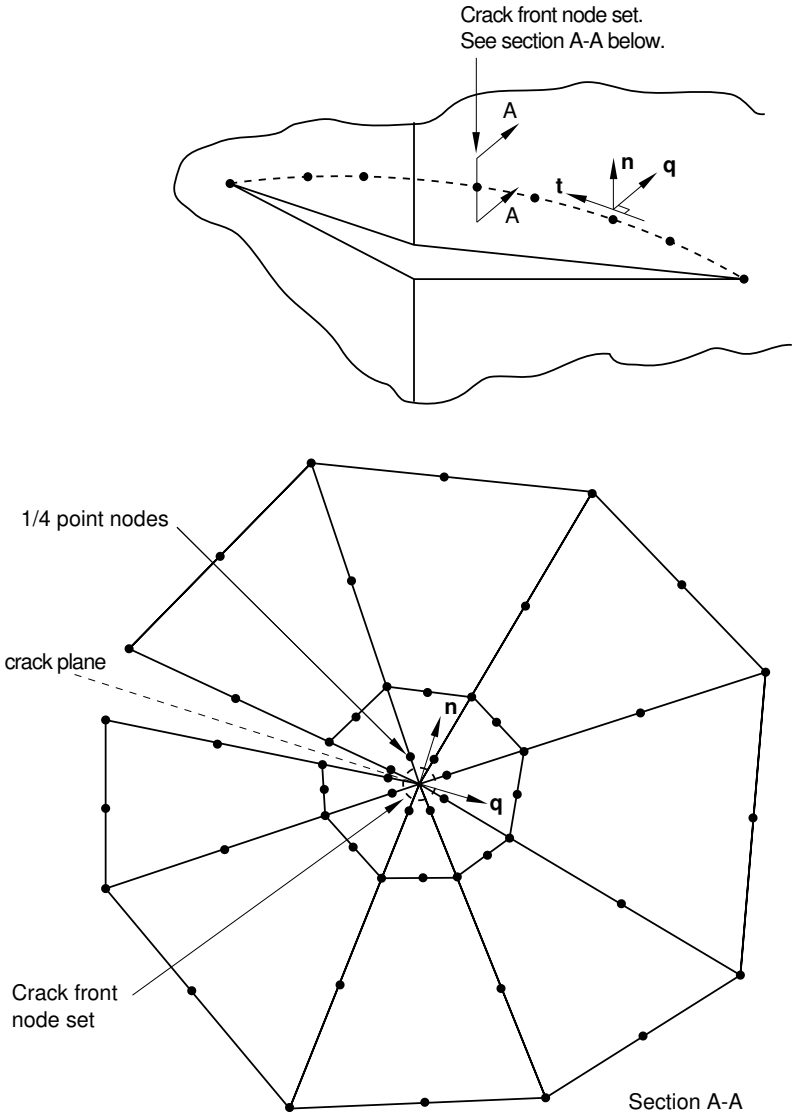


Figure 1: Typical focused mesh for fracture mechanics evaluation.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*, NORMAL

n_x*-direction cosine (or* n_r*),* n_y*-direction cosine (or* n_z*),* n_z*-direction cosine (or blank)*
crack front node set name (2D) or names (3D)

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front: **Specify crack extension direction using: Normal to crack plane**

Specifying the Virtual Crack Extension Direction

Alternatively, the virtual crack extension direction, $\mathbf{q}$, can be specified directly. In three dimensions the virtual crack extension direction, $\mathbf{q}$, will be corrected to be orthogonal to any normal defined at a node or in other cases to the tangent to the crack line itself. The tangent, $\mathbf{t}$, to the crack line at a particular point is obtained by parabolic

interpolation through the crack front for which the virtual crack extension vector is defined and the nearest node sets on either side of this region. Abaqus/Standard will normalize the virtual crack extension direction, $\mathbf{q}$.

Input File Usage: **CONTOUR INTEGRAL*, CONTOURS=*n*
crack front node set name, q_x-direction cosine (or q_r), q_y-direction cosine (or q_z), q_z-direction cosine (or blank)

Repeat the data line for three-dimensional cases to specify the crack front and virtual crack extension vector for each node (or cluster of focused nodes) along the crack line.

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front: **Specify crack extension direction using: q vectors**

Defining Surface Normals

In a case where the crack front intersects the external surface of a three-dimensional solid, where there is a surface of material discontinuity in the model, or where the crack is in a curved shell, the virtual crack extension direction, $\mathbf{q}$, must lie in the plane of the surface for accurate contour integral evaluation. Surface normals should be specified at all nodes that lie on such surfaces within the contours requested for this purpose (these nodes are printed out under the "Contour Integral" information in the data file). For shell element models the normals can be specified with the nodal coordinates if the normals calculated by Abaqus/Standard are not adequate. For solid element models the normals can be specified either directly (see *Normal Definitions at Nodes* and *A plate with a part-through crack: elastic line spring modeling*) or using the nodal coordinates (the fourth–sixth coordinates).

If surface normals are not specified for the nodes on the crack surfaces and the external surfaces at the ends of a crack line, Abaqus/Standard can calculate the normals for these nodes to correct any inadequate virtual crack extension directions, $\mathbf{q}$. For large models, requesting that Abaqus/Standard calculate the surface normals on free or crack surfaces can increase the preprocessing time.

Input File Usage: Use the following option to indicate that no surface normal is calculated:

**CONTOUR INTEGRAL*, SURFACE NORMAL=NONE (default)

Use the following option to calculate the surface normals on the free surfaces:

**CONTOUR INTEGRAL*, SURFACE NORMAL=FREE

Use the following option to calculate the surface normals on the crack surfaces:

**CONTOUR INTEGRAL*, SURFACE NORMAL=CRACK

Use the following option to calculate the surface normals on both the free and crack surfaces:

**CONTOUR INTEGRAL*, SURFACE NORMAL=BOTH

The free surfaces mentioned above denote the external surfaces at the ends of a crack line for a three-dimensional model. They are not relevant for two-dimensional and axisymmetric cases.

Abaqus/CAE Usage: Requesting that Abaqus/Standard calculate the surface normals on the free or crack surfaces is not supported in Abaqus/CAE.

Defining the Data Required for a Contour Integral with XFEM

If you are using XFEM to evaluate the contour integral, both the crack front and the virtual crack extension direction are determined by Abaqus/Standard.

Symmetry with the Conventional Finite Element Method

If the crack is defined on a symmetry plane, only half the structure needs to be modeled. The change in potential energy calculated from the virtual crack front advance is doubled to compute the correct contour integral values.

Input File Usage: Use the following option to indicate that the crack is defined on a symmetry plane:

**CONTOUR INTEGRAL*, CONTOURS=*n*, SYMM

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack tip or crack line, and specify the crack extension direction: **General**: toggle on **On symmetry plane (half-crack model)**

Constructing a Fracture Mechanics Mesh for Small-Strain Analysis with the Conventional Finite Element Method

Sharp cracks (where the crack faces lie on top of one another in the undeformed configuration) are usually modeled using small-strain assumptions. Focused meshes, as shown in *Figure 1*, should typically be used for small-strain fracture mechanics evaluations. However, for a sharp crack the strain field becomes singular at the crack tip. This result is obviously an approximation to the physics; however, the large-strain zone is very localized, and most fracture mechanics problems can be solved satisfactorily using only small-strain analysis.

The crack-tip strain singularity depends on the material model used. Linear elasticity, perfect plasticity, and power-law hardening are commonly used in fracture mechanics analysis. Power-law hardening has the form

$$\frac{\bar{\varepsilon}}{\varepsilon_0} = \alpha\left(\frac{\bar{\sigma}}{\sigma_0}\right)^n,$$

where $\bar{\varepsilon}$ is the equivalent total strain, ε_0 is a reference strain, $\bar{\sigma}$ is the von Mises stress, σ_0 is the initial yield stress, n is the power-law hardening exponent (typically in the range of 3 to 8; $n > 10$ is very close to perfect plasticity for large ε), and α is a material constant (typically in the range 0.5 to 1.0).

Results for pure power-law nonlinear elastic materials in a body under traction loading are proportional to the load to some power. Therefore, the fracture parameters for one geometry under a particular load can be scaled to any other load of the same distribution but different magnitude.

If the loading is proportional (the direction of the stress increase in stress space is approximately constant) and monotonically increasing, power-law hardening deformation plasticity and incremental plasticity are essentially equivalent. However, deformation plasticity is a nonlinear elastic material for which more analytical results are available. Abaqus uses the Ramberg-Osgood form of deformation plasticity (see *Deformation Plasticity*); this model is not a pure power law model, which must be considered.

Creating the Singularity

In most cases the singularity at the crack tip should be considered in small-strain analysis (when geometric nonlinearities are ignored). Including the singularity often improves the accuracy of the *J*-integral, the stress intensity factors, and the stress and strain calculations because the stresses and strains in the region close to the crack tip are more accurate. If r is the distance from the crack tip, the strain singularity in small-strain analysis is

$$\varepsilon \propto r^{-1/2} \quad \text{for linear elasticity,}$$
$$\varepsilon \propto r^{-1} \quad \text{for perfect plasticity, and}$$
$$\varepsilon \propto r^{-\frac{n}{n+1}} \quad \text{for power-law hardening.}$$

Modeling the Crack-Tip Singularity in Two Dimensions

The square root and $1/r$ singularity can be built into a finite element mesh using standard elements. The crack tip is modeled with a ring of collapsed quadrilateral elements, as shown in *Figure 2*.

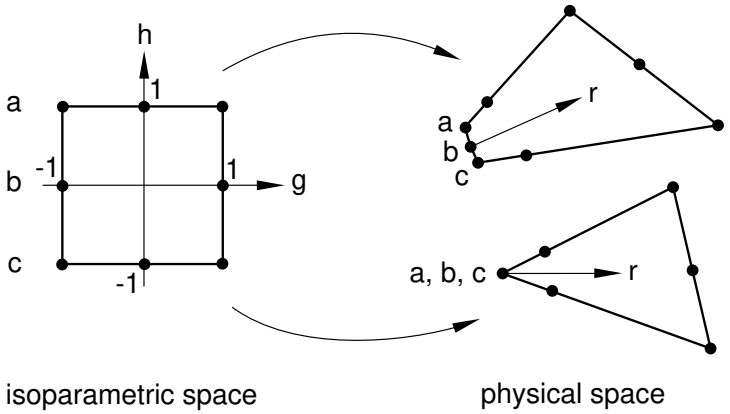


Figure 2: Collapsed two-dimensional element.

To obtain a mesh singularity, generally second-order elements are used and the elements are collapsed as follows:

1. Collapse one side of an 8-node isoparametric element (CPE8R, for example) so that all three nodes—*a*, *b*, and *c*—have the same geometric location (on the crack tip).
2. Move the midside nodes on the sides connected to the crack tip to the 1/4 point nearest the crack tip. You can create "quarter point" spacing with second-order isoparametric elements when you generate nodes for a region of a mesh; see *Creating Quarter-Point Spacing*.

This procedure will create the strain singularity

$$\varepsilon \to \frac{A}{r} + \frac{B}{r^{1/2}} \quad \text{as} \quad r \to 0.$$

The $r^{-\frac{n}{n+1}}$ singularity cannot be created using Abaqus elements, but the combination of the r^{-1} and $r^{-1/2}$ terms can provide a reasonable approximation for $r^{-\frac{n}{n+1}}$.

If 4-node isoparametric elements (for example, CPE4R) are used, one side of the element is collapsed, and the two coincident nodes are free to displace independently, a $1/r$ singularity is created.

If the crack region is meshed with linear elements, the position specified for the midside nodes is ignored.

Creating a Square Root Singularity

If nodes *a*, *b*, and *c* are constrained to move together, $A = 0$ and the strains and stresses are square root singular (suitable for linear elasticity).

Input File Usage: **NFILL*, SINGULAR

Constrain the collapsed nodes to move together by specifying the same node number in the list of nodes forming the element or by using a linear constraint equation or multi-point constraint to tie them together.

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, single node**

Creating a *1/r* Singularity

If the midside nodes remain at the midside points rather than being moved to the 1/4 points and nodes *a*, *b*, and *c* are allowed to move independently, only the $1/r$ singularity in strain is created (suitable for perfect plasticity).

Input File Usage: **NFILL*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.5, **Collapsed element side, duplicate nodes**

Creating a Combined Square Root and *1/r* Singularity

If the midside nodes are moved to the 1/4 points but nodes *a*, *b*, and *c* are allowed to move independently, the singularity created is a combination of the square root and $1/r$ singularities. This combination is usually best for a power-law hardening material. However, since the $1/r$ singularity dominates, moving the midside nodes to the 1/4 points gives only slightly better results than if the nodes are left at the midside points. Since creating a mesh with the midside nodes moved to the quarter points can be difficult, it is often best to simply use the $1/r$ singularity.

Input File Usage: **NFILL*, SINGULAR

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, duplicate nodes**

Modeling the Crack-Tip Singularity in Three Dimensions

To create singular fields, 20-node bricks and 27-node bricks can be used with a collapsed face (see *Figure 3*).

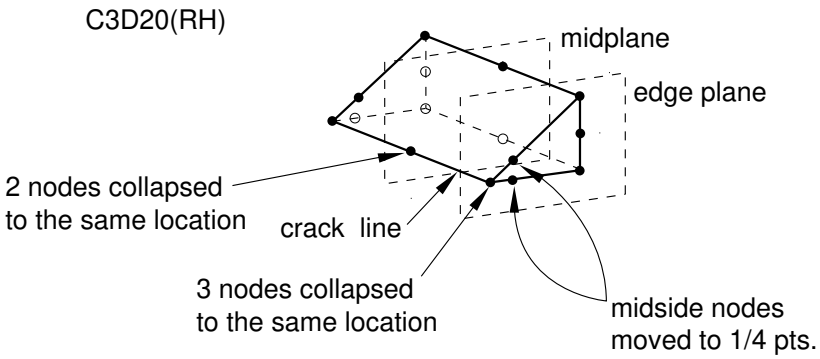


Figure 3: Collapsed three-dimensional element.

The planes of the three-dimensional elements perpendicular to the crack line should be planar for the best accuracy. If they are not planar, the element Jacobian might become negative at some integration points when the midside nodes are moved to the 1/4 points. To correct this problem, move the midside nodes slightly away from the 1/4 points toward the midpoint position (the distance moved is not critical).

See *Meshing the crack region and assigning elements* for information on creating a three-dimensional fracture mechanics mesh in Abaqus/CAE.

Creating a Square Root Singularity

To obtain a square root singularity, constrain the nodes on the collapsed face of the edge planes to move together and move the nodes to the 1/4 points.

If the nodes at the midplane of a collapsed 20-node brick are constrained to move together, $\varepsilon \propto r$; therefore, the singularity is not the same on the midplane as on an edge plane. This difference causes local oscillations in the solution about the crack tip along the crack line, although normally the oscillations are not significant.

If all midface nodes and the centroid node are included in a 27-node brick and the midside and midface nodes are moved to the 1/4 points closest to the crack line, the oscillation in the local stress and strain fields can be reduced.

Input File Usage: **NFILL*, SINGULAR

Constrain the collapsed nodes to move together by specifying the same node number in the list of nodes forming the element or by using a linear constraint equation or multi-point constraint to tie them together.

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, single node**

Creating a *1/r* Singularity

To obtain a $1/r$ singularity, allow the three nodes on the collapsed face to displace independently and keep the midside nodes at the midpoints.

Input File Usage: **NFILL*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.5, **Collapsed element side, duplicate nodes**

Creating a Combined Square Root and *1/r* Singularity

To obtain a combined square root and $1/r$ singularity, allow the nodes on the collapsed face to displace independently and move the midside nodes to the 1/4 points. As in the two-dimensional case, if it is difficult to create the mesh with the nodes moved to the 1/4 points, simply use the $1/r$ singularity.

Input File Usage: **NFILL*, SINGULAR

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, duplicate nodes**

Mesh Refinement

The size of the crack-tip elements influences the accuracy of the solutions: the smaller the radial dimension of the elements from the crack tip, the better the stress, strain, etc. results will be and, therefore, the better the contour integral calculations will be.

The angular strain dependence is not modeled with the singular elements. Reasonable results are obtained if typical elements around the crack tip subtend angles in the range of 10° (accurate) to 22.5° (moderately accurate).

Since the crack tip causes a stress concentration, the stress and strain gradients are large as the crack tip is approached. Path dependence in the evaluation of the *J*-integral might be an indication that the mesh is not sufficiently refined, but path independence does not prove mesh convergence. The finite element mesh must be refined in the vicinity of the crack to get accurate stresses and strains; however, accurate *J*-integral results can frequently be obtained even with a relatively coarse mesh.

In many cases if sufficiently fine meshes are used, accurate contour integral values can be obtained without using singular elements.

Modeling the Crack-Tip Region in Shells

Focused meshes can be used, but not all the three-dimensional shell elements in Abaqus/Standard can be collapsed. Elements S8R and S8RT cannot be degenerated into triangles; element types S4, S4R, S4R5, S8R5, and S9R5 can.

The quarter-point technique (moving the midside nodes to the quarter points to give a $1/\sqrt{r}$ singularity for elastic fracture mechanics applications) can be used with S8R5 and S9R5 elements but not with S8R(T) elements. When the quarter-point technique is used with S9R5 elements, the midface node should be moved to the quarter-point position along with the two midside nodes.

If S8R(T) elements are used, a keyhole should be introduced at the crack tip.

Flaws lying in the plane through the thickness of a shell can be modeled using line spring elements; see *Line Spring Elements for Modeling Part-through Cracks in Shells*. In many cases line spring elements provide accurate *J*-integral and stress intensity values, but these elements are limited to modeling small strain and rotations. Limited modeling of plasticity is also allowed with line springs.

Constructing a Fracture Mechanics Mesh for Finite-Strain Analysis with the Conventional Finite Element Method

In large-strain analysis (when geometric nonlinearities are included) singular elements should not normally be used. The mesh must be sufficiently refined to model the very high strain gradients around the crack tip if details in this region are required. Even if only the *J*-integral is required, the deformation around the crack tip might dominate the solution and the crack-tip region will have to be modeled with sufficient detail to avoid numerical problems.

Physically, the crack tip is not perfectly sharp. Therefore, it is normally modeled as a blunted notch with a radius of $\sim 10^{-3} r_p$, where r_p is a characteristic dimension of the plastic zone ahead of the crack tip. The notch must be small enough that, at the loads of interest, the deformed shape of the notch no longer depends on the original geometry. Typically, the notch must blunt out to more than four times its original radius for the deformed shape to be independent of the original geometry. The size of the elements around the notch should be about 1/10 the notch-tip radius to obtain accurate results.

If a crack is modeled as sharp, the finite elements near the crack tip might not be able to approximate the high gradients, resulting in convergence problems. The stress and strain results around the crack tip will probably be inaccurate even if convergence is achieved. However, if the solution converges, the contour integral results should be reasonably accurate. The convergence difficulties will probably be greater in three dimensions than in two dimensions.

In situations involving finite rotations but small strains, such as bending of slender structures, a small “keyhole” around the crack tip should be modeled. If the hole is small, the results will not be affected significantly and problems in dealing with the singular strains at the crack tip will be avoided.

Using Constraints with the Conventional Finite Element Method

General multi-point constraints and linear constraint equations (*About Kinematic Constraints*) should not be used on nodes in the mesh regions where contour integrals are calculated unless the nodes involved in the constraint are located at the same point. The nodes at the crack tip of a focused mesh can be tied together using multi-point constraints without adversely affecting the contour integral calculations. Tying these nodes will change the singularity at the crack tip, but path independence of the contour integral will be maintained. In addition, path independence of the contour integrals will not be affected if two faces of a model are joined using MPC type TIE or a linear constraint equation, provided that all nodes of the two faces are coincident. Using multi-point constraints for mesh refinement or for applying symmetry/antisymmetry boundary conditions within the contour integral region will result in path dependence of the contour integrals. No warning or error messages are provided if this rule is violated.

Procedures

You can request contour integrals in fracture mechanics problems that were modeled using the following procedures:

- static (*Static Stress Analysis*) with both XFEM and the conventional finite element methods;
- quasi-static (*Quasi-Static Analysis*) with the conventional finite element method only;
- steady-state transport (*Steady-State Transport Analysis*) with the conventional finite element method only;
- coupled thermal-stress procedures (*Fully Coupled Thermal-Stress Analysis*) with the conventional finite element method only;
- crack propagation (*Crack Propagation Analysis*) with the conventional finite element method only; and
- linear elastic fatigue crack growth analysis (*Linear Elastic Fatigue Crack Growth Analysis*) with the conventional finite element method only.

Contour integrals can be requested only in general analysis steps: they are not calculated in linear perturbation analyses (*General and Perturbation Procedures*).

A crack analysis with pressure applied on the crack surfaces might give inaccurate contour integral values if geometric nonlinearity is included in a step. Similarly, the calculated results of the stress intensity factors and T-stress might not be accurate if geometric nonlinearity is included in a step.

Loads

Contour integral calculations include the following distributed load types:

- thermal loads;
- distributed loads, including crack face pressure and traction loads on continuum elements as well as those applied using user subroutine *DLOAD* and *UTRACLOAD*;
- distributed loads, including surface traction loads and crack face edge loads on shell elements as well as those applied using user subroutine *UTRACLOAD*;
- uniform and nonuniform body forces; and
- centrifugal loads on continuum and shell elements.

Contributions to the contour integral due to concentrated loads in the domain are not included; instead, the mesh must be modified to include a small element and a distributed load must be applied to this element.

Contributions due to contact forces are not included.

Material Options

J-integral calculations are valid for linear elastic (including isotropic, orthotropic, or anisotropic material behavior), nonlinear elastic, and elastic-plastic materials . Plastic behavior can be modeled as nonlinear elastic (*Deformation Plasticity*), but the results are generally best if the material is modeled by incremental plasticity and is subject to proportional, monotonic traction loading.

If unloading has taken place in the plastic zone around the crack tip, the *J*-integral will not be valid except in very limited cases.

The C_t-integral is valid for problems involving creep (*Rate-Dependent Plasticity: Creep and Swelling*).

The stress intensity factor calculation is valid for cracks in linear elastic materials including isotropic, orthotropic, or anisotropic material behavior with small elastic strains. It is also valid for an interfacial crack between two different isotropic linear elastic materials. It is not valid for any other types of materials, including user-defined materials.

The crack propagation direction is valid only for homogeneous, isotropic linear elastic materials.

The *T*-stress is valid only for homogeneous, isotropic linear elastic materials. Although the *T*-stress is calculated using the linear elastic material properties of the body with a crack, it is usually used with the *J*-integral calculated using the elastic-plastic material properties of the body (see *T-stress extraction*).

If there is material discontinuity, the normal to the material discontinuity line must be specified for all nodes on the material discontinuity that will lie in a contour integral domain. The normal can be specified by defining user-specified normals (see *Normal Definitions at Nodes*) for the elements on both sides of the discontinuity or by using nodal normal coordinates for the nodes on the discontinuity. Contour integral calculations cannot be performed for a crack with a material discontinuity line passing through its tip (except for an interfacial crack between two different materials). Therefore, you should be careful when specifying a normal that is not perpendicular to the virtual crack extension direction, $\mathbf{q}$, for the nodes at the crack tip.

Elements

When used with XFEM, the contour integral can be evaluated only in first-order or second-order tetrahedral and first-order brick elements. The following paragraphs apply only to the conventional finite element method.

The contour integral evaluation capability in Abaqus/Standard assumes that the elements that lie within the domain used for the calculations are quadrilateral or triangular elements in two-dimensional or shell models or are bricks or second-order tetrahedral elements in continuum three-dimensional models. You should not use wedges in the mesh that is included in the contour integral regions. When the elements around the crack tip are generated in Abaqus/CAE, wedge elements are converted to collapsed hexahedral elements. The elements within the contour domain should be of the same type.

In shell structures the contour integrals calculated by Abaqus/Standard are contour independent only if the deformation mode around the crack tip is primarily membrane. If there are significant bending or transverse shear effects in the domain, the contour integrals might not be contour independent and contour integral values should be obtained directly from the displacements and/or the stresses.

Generalized plane strain elements, generalized axisymmetric elements with twist, asymmetric-axisymmetric elements, membrane elements, and cylindrical elements should not be used in the contour integral regions.

The contribution of rebar is included only in the calculations of the *J*-integral and the C_t-integral for shell elements defined with a shell section integrated during the analysis (see *Using a Shell Section Integrated during the Analysis to Define the Section Behavior*).

Output

The domain associated with each contour is calculated automatically. The nodes belonging to each domain can be printed in the data file; see *Controlling the Amount of analysis input file processor Information Written to the Data File*. If you are using the conventional contour integral method, for each domain Abaqus/Standard creates a new node set in the output database to include these nodes; you can view these node sets in Abaqus/CAE. In addition, new node sets are created in the output database for nodes on crack surfaces and on free surfaces whose nodal normals are calculated by Abaqus/Standard.

Contour integrals cannot be recovered from the restart file as described in *About Output*.

You should not request element output extrapolated to the nodes (*Element Output*) for second-order elements with one collapsed side in two dimensions or one collapsed face in three dimensions.

Default Contour Integral Output

By default, the contour integral values are written to the data file and to the output database file. The following naming convention is used for contour integrals written to the output database:

integral-type`:` *abbrev-integral-type* `at`
history-output-request-name`_`*crack-name*`_`*internal-crack-tip-node-set-name*`__Contour_`*contour-number*

where *integral-type* can be

- `Crack propagation direction (Cpd)`
- `J-integral (J)`
- `J-integral estimated from Ks (JKs)`
- `Stress intensity factor K1 (K1)`
- `Stress intensity factor K2 (K2)`
- `T-stress (T)`

For example,

```
J-integral: J at JINT_CRACK_CRACKTIP-1__Contour_1
```

Writing the Contour Integrals to the Results File

You can choose to write the contour integral values to the results file in addition to the data file.

Input File Usage: Use the following option to write the contour integrals to the results file instead of the data file:

**CONTOUR INTEGRAL*, CONTOURS=*n*, OUTPUT=FILE

Use the following option to write the contour integrals to the results file in addition to the data file:

**CONTOUR INTEGRAL*, CONTOURS=*n*, OUTPUT=BOTH

Abaqus/CAE Usage: You cannot write contour integrals to the results file from Abaqus/CAE.

Controlling the Output Frequency

You can control the output frequency, in increments, of contour integrals. By default, the crack-tip location and associated quantities will be printed every increment. Specify an output frequency of 0 to suppress contour integral output.

The output frequency for contour integral output to the output database is controlled by the larger of the frequency values specified for history output to the output database (see *Output to the Output Database*) or for contour integral output. If you specify an output frequency of 0 for the history output to the output database, contour integral values will not be written to the output database.

Input File Usage: **CONTOUR INTEGRAL*, CRACK NAME=*crack name*, CONTOURS=*n*, FREQUENCY=*f*

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Save output at:** ***f***

Requesting Field Output of the Contour Integral

For the conventional contour integral method, if n contours are specified, you can request the averaged value of contour integrals over $\frac{n}{2}$ contours (starting from contour number $\frac{n}{2}$) as nodal field output written to the output database.

Nodal field output is not available for the XFEM-based contour integral method.

The following nodal field output variables are available:

J	Averaged value of the contour integral.
K	Averaged values of the stress intensity factors.
TSTRESS	Averaged value of the T-stress.

Input File Usage: **NODE OUTPUT*

Abaqus/CAE Usage: Step module: field output request editor

References

- Lei, Y., "J-Integral Evaluation for Cases Involving Non-proportional Stressing," *Engineering Fracture Mechanics*, vol. 72, pp. 577–596, 2005.
- Lei, Y., N. P. O'Dowd, and G. A. Webster, "Fracture Mechanics Analysis of a Crack in a Residual Stress Field," *International Journal of Fracture*, vol. 106, pp. 195–216, 2000.

Crack Propagation Analysis

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Defining an Analysis*
- *About Fracture Mechanics*
- *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*
- *Linear Elastic Fatigue Crack Growth Analysis*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- *Contact Cohesive Behavior*
- **SURFACE INTERACTION*
- **COHESIVE BEHAVIOR*
- **CONTACT CLEARANCE*
- **DEBOND*
- **DIRECT CYCLIC*
- **FATIGUE*
- **FRACTURE CRITERION*
- **NODAL ENERGY RATE*
- *Defining surface-to-surface contact in an Abaqus/Standard analysis*

Overview

Crack propagation analysis:

- allows for numerous types of fracture criteria in Abaqus/Standard—critical stress at a certain distance ahead of the crack tip, critical crack opening displacement, crack length versus time, VCCT (the Virtual Crack Closure Technique), enhanced VCCT, PRX-VCCT (the Progressive Release eXplicit - Virtual Crack Closure Technique), and the fatigue crack growth criterion based on the Paris law (utilizing either the traditional VCCT or the PRX-VCCT technique to calculate the fracture energy release rate);
- allows for the VCCT fracture criterion in Abaqus/Explicit;
- in Abaqus/Standard models quasi-static crack growth in two dimensions (planar and axisymmetric) for all types of fracture criteria and in three dimensions (solid, shells, and continuum shells) for VCCT, enhanced VCCT, PRX-VCCT, and the fatigue crack growth criteria (utilizing either the traditional VCCT or the PRX-VCCT technique to calculate the fracture energy release rate);
- in Abaqus/Explicit models crack growth in two dimensions (planar and axisymmetric) and in three dimensions (solid, shells, and continuum shells) for the VCCT criterion; and
- requires that you define two distinct initially bonded contact surfaces between which the crack will propagate.

Defining Initially Bonded Crack Surfaces in Abaqus/Standard

Potential crack surfaces are modeled as secondary and main contact surfaces (see *About Contact Pairs in Abaqus/Standard*). Any contact formulation except the finite-sliding, surface-to-surface formulation can be used. The predetermined crack surfaces are assumed to be initially partially bonded so that the crack tips can be identified explicitly by Abaqus/Standard. Initially bonded crack surfaces cannot be used with self-contact.

Define an initial condition to identify which part of the crack is initially bonded. You specify the secondary surface, the main surface, and a node set that identifies the initially bonded part of the secondary surface. The unbonded portion of the secondary surface will behave as a regular contact surface. Either the secondary surface or the main surface must be specified; if only the main surface is given, all the secondary surfaces associated with this main surface that have nodes in the node set will be bonded at these nodes.

If a node set is not specified, the initial contact conditions will apply to the entire contact pair; in this case, no crack tips can be identified, and the bonded surfaces cannot separate.

If a node set is specified, the initial conditions apply only to the secondary nodes in the node set. Abaqus/Standard checks to ensure that the node set defined includes only secondary nodes belonging to the contact pair specified.

By default, the nodes in the node set are considered to be initially bonded in all directions.

Input File Usage: **INITIAL CONDITIONS*, TYPE=CONTACT

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)**

Bonding Only in the Normal Direction

For fracture criteria based on the critical stress, critical crack opening displacement, or crack length versus time, it is possible to bond the nodes in the node set (or the contact pair if a node set is not defined) only in the normal direction. In this case the nodes are allowed to move freely tangential to the contact surfaces. Friction (*Frictional Behavior*) cannot be specified if the nodes are bonded only in the normal direction.

Bonding only in the normal direction is typically used to model bonded contact conditions in Mode I crack problems where the shear stress ahead of the crack along the crack plane is zero.

Input File Usage: **INITIAL CONDITIONS*, TYPE=CONTACT, NORMAL

Abaqus/CAE Usage: Bonding only in the normal direction is not supported in Abaqus/CAE.

Activating the Crack Propagation Capability in Abaqus/Standard

The crack propagation capability must be activated within the step definition to specify that crack propagation might occur between the two surfaces that are initially partially bonded. You specify the surfaces along which the crack propagates.

If the crack propagation capability is not activated for partially bonded surfaces, the surfaces will not separate; in this case the specified initial contact conditions would have the same effect as that provided by the tied contact capability, which generates a permanent bond between two surfaces during the entire analysis (see *Defining Tied Contact in Abaqus/Standard*).

Input File Usage: **SURFACE INTERACTION*, NAME=*interaction_name*
**FRACTURE CRITERION*
**
**DEBOND*, SECONDARY=*secondary_surface_name*, MAIN=*main_surface_name*, INTERACTION=*interaction_name*

Abaqus/CAE Usage: The debond feature cannot be associated with a surface interaction definition in Abaqus/CAE. However, the surface interaction and fracture criterion options can be defined in Abaqus/CAE.

Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)**, select main and secondary surfaces

Propagation of Multiple Cracks

Cracks can propagate from either a single crack tip or multiple crack tips. The crack propagation capability in Abaqus/Standard requires that the surfaces be initially partially bonded so that the crack tips can be identified. A contact pair can have crack propagation from multiple crack tips. However, only one crack propagation criterion is allowed for a given contact pair. Crack propagation along several contact pairs can be modeled by specifying multiple crack propagation definitions.

Defining and Activating Crack Propagation in Abaqus/Explicit

In Abaqus/Explicit potential crack surfaces are modeled as bonded general contact surfaces (see *About General Contact in Abaqus/Explicit*) in the context of surface-based cohesive behavior (see *Contact Cohesive Behavior*). The capability is implemented using a pure main-secondary formulation. As is the case in Abaqus/Standard, the predetermined crack surfaces are assumed to be initially partially bonded so that the crack tips can be identified explicitly.

To identify which pair of surfaces determine the crack and which part of the crack is initially bonded, you must define and assign a contact clearance (see *Contact Initialization for General Contact in Abaqus/Explicit*). You first define a contact clearance to specify the node set that is initially bonded, and then you assign this contact clearance to a pair of two single-sided surfaces that define the crack. The unbonded portion behaves as a regular contact surface. The nodes in the node set are considered to be initially bonded in all directions.

The crack tip is identified only from the specified two surfaces and the node set. No attempt is made to determine a crack tip from all surfaces included in the general contact domain. Consequently, to be able to identify the crack tip, the surface including the specified node set must extend past the node set. Otherwise, the surfaces will not debond, and the crack cannot propagate.

You complete the definition of the crack propagation capability by defining a fracture-based cohesive behavior surface interaction. You activate the crack propagation by assigning it to the pair of surfaces that are initially partially bonded. If the fracture criterion is met, crack propagation occurs between these two surfaces. Cohesive behavior is also used to specify the elastic behavior of the bonds (see *Contact Cohesive Behavior*).

If a fracture-based surface interaction is not assigned to a pair of surfaces, the crack definition is incomplete. Unlike Abaqus/Standard where the identified nodes will stay bonded if the crack is not activated, in Abaqus/Explicit the nodes identified by the contact clearance definition will separate without generating any interface stress.

Similar to Abaqus/Standard, cracks can propagate from single or multiple crack tips for the same pair of surfaces.

Input File Usage: Use the following options:

**CONTACT CLEARANCE*, NAME=*clearance_name*, SEARCH
NSET=*bonded_nset_name*
**
**SURFACE INTERACTION*, NAME=*interaction_name*
**COHESIVE BEHAVIOR*
**FRACTURE CRITERION*
..**
**CONTACT*
**CONTACT CLEARANCE ASSIGNMENT*
secondary_surface, *main_surface*, *clearance_name*
**CONTACT PROPERTY ASSIGNMENT*
secondary_surface, *main_surface*, *interaction_name*

Abaqus/CAE Usage: Defining and activating crack propagation in Abaqus/Explicitis not supported in Abaqus/CAE.

Specifying a Fracture Criterion

You can specify the crack propagation criteria, as discussed below. *Table 1* shows which criteria are supported by Abaqus/Standard and Abaqus/Explicit. Only one crack propagation criterion is allowed per contact pair even if multiple cracks are present.

Table 1: Available fracture criterion.

Crack propagation criterion	Abaqus/Standard	Abaqus/Explicit
Critical stress	Yes	No
Critical crack opening displacement	Yes	No
Crack length versus time	Yes	No
VCCT	Yes	Yes
Enhanced VCCT	Yes	No
PRX-VCCT	Yes	No
Fatigue crack growth utilizing traditional VCCT to calculate the fracture energy release rate	Yes	No
Fatigue crack growth utilizing PRX-VCCT to calculate the fracture energy release rate	Yes	No

Crack propagation analysis is carried out on a nodal basis. The crack-tip node debonds when the fracture criterion, f, reaches the value 1.0 within a given tolerance:

$$f_{LL} \le f \le f_{UL},$$

where $f_{UL} = 1 + f_{tol}$ and $f_{LL} = 1$ for VCCT, enhanced VCCT, PRX-VCCT, and fatigue crack growth criteria or $f_{LL} = 1 - f_{tol}$ for other fracture criteria. You can specify the tolerance f_{tol}. In Abaqus/Standard, if $f > 1 + f_{tol}$, the time increment is cut back such that the crack propagation criterion is satisfied except in the case of an unstable crack growth problem where multiple nodes at and ahead of a crack tip are allowed to debond without the cut back of increment size in one increment. The default value of f_{tol} is 0.1 for the critical stress, critical crack opening displacement, and crack length versus time criteria and is 0.2 for the VCCT, enhanced VCCT, and PRX-VCCT criteria.

Input File Usage: **FRACTURE CRITERION*, TOLERANCE=f_{tol}, TYPE=*type*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical**->**Fracture Criterion**, **Type**: **VCCT** or **Enhanced VCCT**, **Tolerance**

Critical Stress Criterion

This criterion is available only in Abaqus/Standard.

If you specify a critical stress criterion at a critical distance ahead of the crack tip, the crack-tip node debonds when the local stress across the interface at a specified distance ahead of the crack tip reaches a critical value.

This criterion is typically used for crack propagation in brittle materials. The critical stress criterion is defined as

$$f = \sqrt{\left(\frac{\hat{\sigma}_n}{\sigma^f}\right)^2 + \left(\frac{\tau_1}{\tau_1^f}\right)^2 + \left(\frac{\tau_2}{\tau_2^f}\right)^2}, \quad \hat{\sigma}_n = \max(\sigma_n, 0),$$

where σ_n is the normal component of stress carried across the interface at the distance specified; τ_1 and τ_2 are the shear stress components in the interface; and σ^f and τ_1^f are the normal and shear failure stresses, which you must specify. The second component of the shear failure stress, τ_2, is not relevant in a two-dimensional analysis; therefore, the value of τ_2^f need not be specified. The crack-tip node debonds when the fracture criterion, f, reaches the value 1.0.

If the value of τ_1^f is not given or is specified as zero, it will be taken to be a very large number so that the shear stress has no effect on the fracture criterion.

The distance ahead of the crack tip is measured along the secondary surface, as shown in *Figure 1*. The stresses at the specified distance ahead of the crack tip are obtained by interpolating the values at the adjacent nodes. The interpolation depends on whether first-order or second-order elements are used to define the secondary surface.

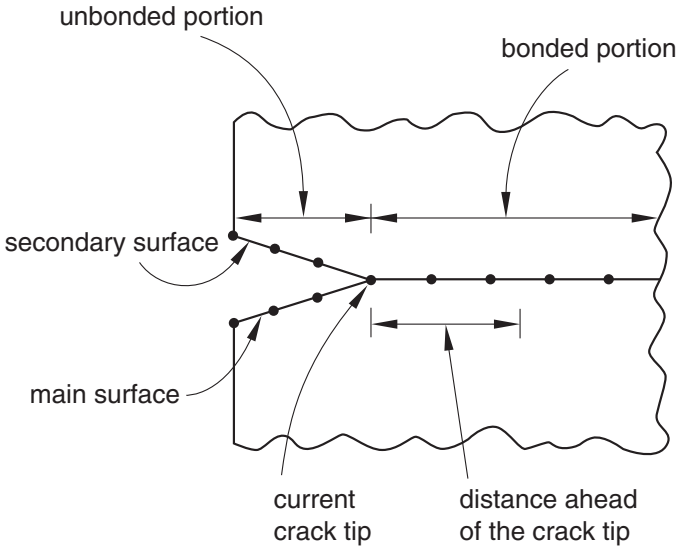


Figure 1: Distance specification for the critical stress criterion.

Input File Usage: **FRACTURE CRITERION*, TYPE=CRITICAL STRESS, DISTANCE=*n*

Abaqus/CAE Usage: The critical stress criterion is not supported in Abaqus/CAE.

Critical Crack Opening Displacement Criterion

This criterion is available only in Abaqus/Standard.

If you base the crack propagation analysis on the crack opening displacement criterion, the crack-tip node debonds when the crack opening displacement at a specified distance behind the crack tip reaches a critical value. This criterion is typically used for crack propagation in ductile materials.

The crack opening displacement criterion is defined as

$$f = \frac{\delta}{\delta_c},$$

where δ is the measured value of crack opening displacement and δ_c is the critical value of the crack opening displacement (user-specified). The crack-tip node debonds when the fracture criterion reaches the value 1.0.

You must supply the crack opening displacement versus cumulative crack length data. In Abaqus/Standard the cumulative crack length is defined as the distance between the initial crack tip and the current crack tip measured along the secondary surface in the current configuration. The crack opening displacement is defined as the normal distance separating the two faces of the crack at the given distance.

You specify the position, *n*, behind the crack tip where the critical crack opening displacement is calculated. The value of this position must be specified as the length of the straight line joining the current crack tip and points on the secondary and main surfaces (*Figure 2*).

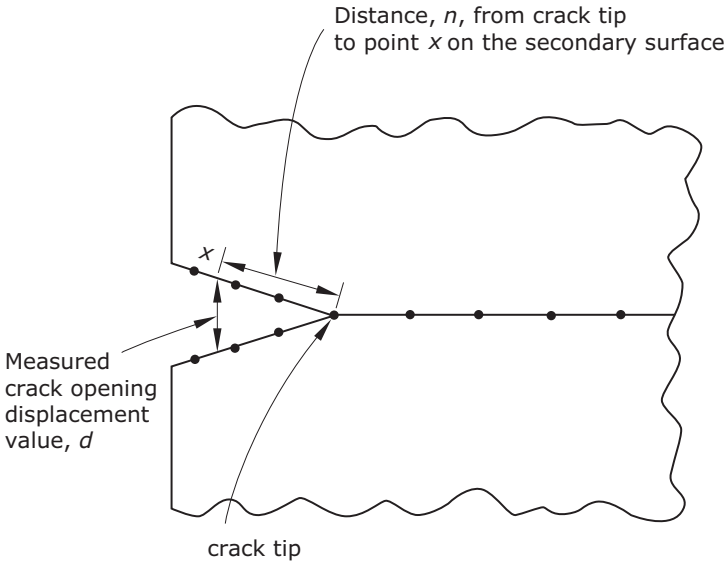


Figure 2: Distance specification for the critical crack opening displacement criterion.

Abaqus/Standard computes the crack opening displacement at that point by interpolating the values at the adjacent nodes. The interpolation depends on whether first-order or second-order elements are used to define the secondary surface. An error message will be issued if the value of *n* is not within the end points of the contact pair.

Input File Usage: **FRACTURE CRITERION*, TYPE=COD, DISTANCE=*n*

Abaqus/CAE Usage: The critical crack opening displacement criterion is not supported in Abaqus/CAE.

Modeling Symmetry

In problems where the debonding surfaces lie on a symmetry plane, you can specify that Abaqus/Standard should consider only half of the user-specified crack opening displacement values. In this case the initial bonding must be in the normal direction only (see *Bonding Only in the Normal Direction* above).

Input File Usage: **FRACTURE CRITERION*, TYPE=COD, DISTANCE=*n*, SYMMETRY

Abaqus/CAE Usage: Modeling symmetry is not supported in Abaqus/CAE.

Crack Length Versus Time Criterion

This criterion is available only in Abaqus/Standard.

To specify the crack propagation explicitly as a function of total time, you must provide a crack length versus time relationship and a reference point from which the crack length is measured. This reference point is defined by specifying a node set. Abaqus/Standard finds the average of the current positions of the nodes in the set to define the reference point. During crack propagation the crack length is measured from this user-specified reference point along the secondary surface in the deformed configuration. The time specified must be total time, not step time.

The fracture criterion, f, is stated in terms of the user-specified crack length and the length of the current crack tip. The length of the current crack tip from the reference point is measured as the sum of the straight line distance of the initial crack tip from the reference point and the distance between the initial crack tip and the current crack tip measured along the secondary surface.

Referring to *Figure 3*, let node 1 be the initial location of the crack tip and node 3 be the current location of the crack tip. The distance of the current crack tip located at node 3 is given by

$$l_3 = l_1 + \Delta l_{12} + \Delta l_{23},$$

where l_1 is the length of the straight line joining node 1 and the reference point, Δl_{12} is the distance between nodes 1 and 2, and Δl_{23} is the distance between nodes 2 and 3 measured along the secondary surface.

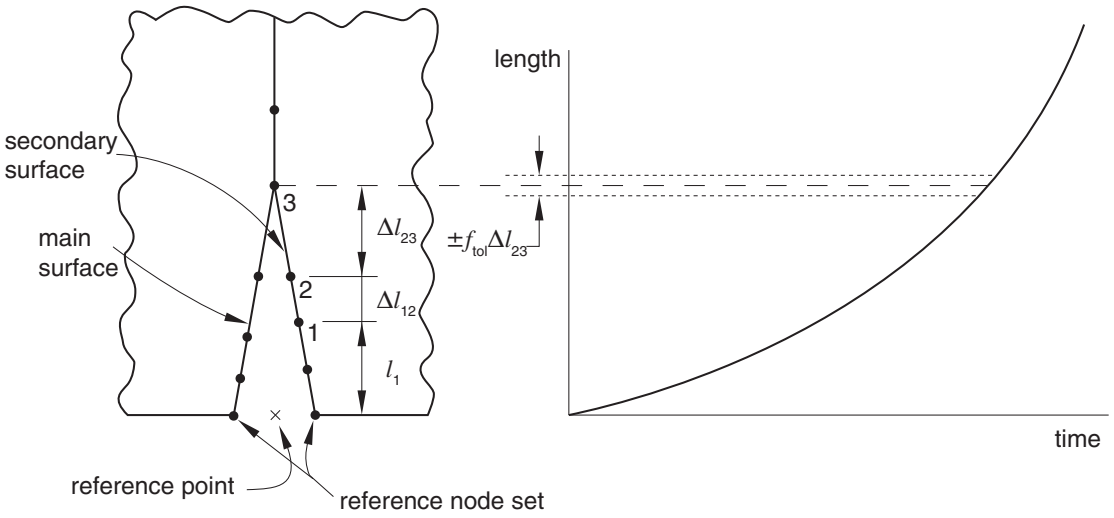


Figure 3: Crack propagation as a function of time.

The fracture criterion, f, is given by

$$f = \frac{l-(l_3-\Delta l_{23})}{\Delta l_{23}},$$

where l is the length at the current time obtained from the user-specified crack length versus time curve. Crack-tip node 3 will debond when the failure function f reaches the value of 1.0 (within the user-defined tolerance).

If geometric nonlinearity is considered in the step (*Defining an Analysis*), the reference point might move as the body deforms; you must ensure that this movement does not invalidate the crack length versus time criterion.

Abaqus/Standard does not extrapolate beyond the end points of your crack data. Therefore, if the first crack length specified is greater than the distance from the crack reference point to the first bonded node, the first bonded node will never debond and the crack will not propagate. In this case Abaqus/Standard will print warning messages in the message (`.msg`) file.

Input File Usage: **FRACTURE CRITERION*, TYPE=CRACK LENGTH, NSET=*name*

Abaqus/CAE Usage: The crack length versus time criterion is not supported in Abaqus/CAE.

VCCT Criterion

This criterion is available in both Abaqus/Standard and Abaqus/Explicit.

The Virtual Crack Closure Technique (VCCT) criterion uses the principles of linear elastic fracture mechanics (LEFM) and, therefore, is appropriate for problems in which brittle crack propagation occurs along predefined surfaces.

VCCT is based on the assumption that the strain energy released when a crack is extended by a certain amount is the same as the energy required to close the crack by the same amount. For example, *Figure 4* illustrates the similarity between crack extension from i to j and crack closure at j.

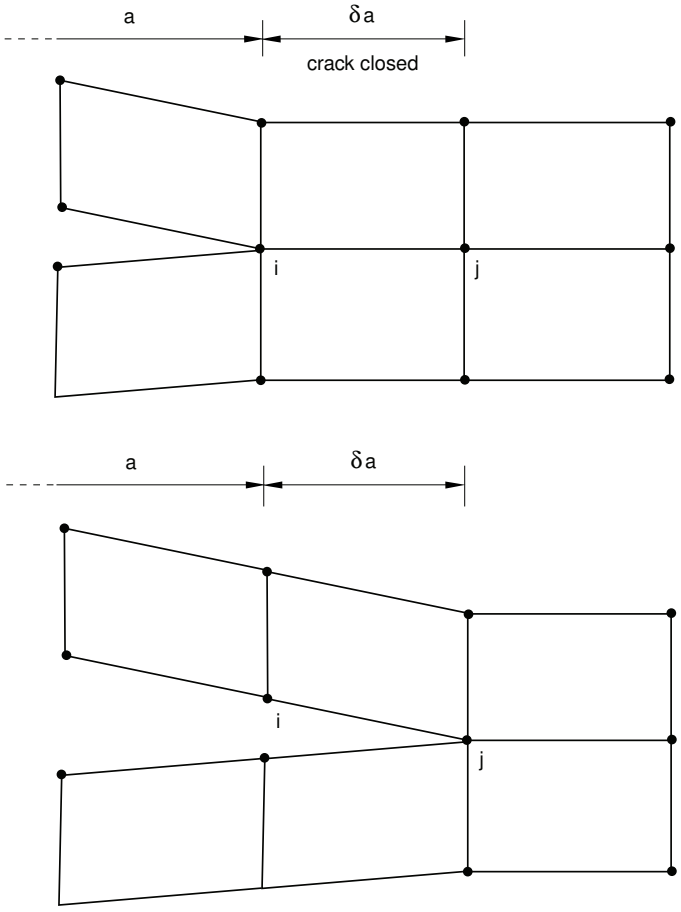


Figure 4: Mode I: The energy released when a crack is extended by a certain amount is the same as the energy required to close the crack.

In *Figure 5* nodes 2 and 5 will start to release when

$$f = \frac{G_I}{G_{IC}} = \frac{1}{2}\left(\frac{v_{1,6}F_{v,2,5}}{bd}\right)\frac{1}{G_{IC}} \geq 1.0,$$

where G_I is the Mode I energy release rate, G_{IC} is the critical Mode I energy release rate, b is the width, d is the length of the elements at the crack front, $F_{v,2,5}$ is the vertical force between nodes 2 and 5, and $v_{1,6}$ is the vertical displacement between nodes 1 and 6. Assuming that the crack closure is governed by linear elastic behavior, the energy to close the crack (and, thus, the energy to open the crack) is calculated from the previous equation. Similar arguments and equations can be written in two dimensions for Mode II and for three-dimensional crack surfaces including Mode III.

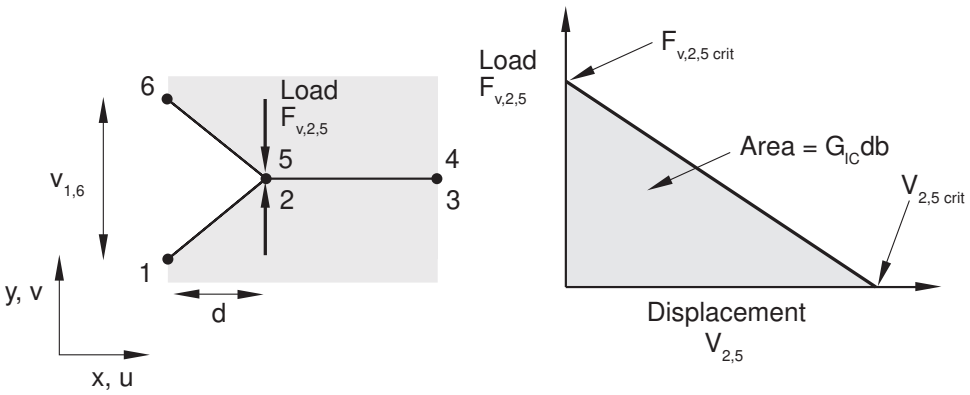


Figure 5: Pure Mode I modified.

In the general case involving Mode I, II, and III the fracture criterion is defined as

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0,$$

where G_{equiv} is the equivalent strain energy release rate calculated at a node, and G_{equivC} is the critical equivalent strain energy release rate calculated based on the user-specified mode mix criterion and the bond strength of the interface. The crack-tip node will debond when the fracture criterion reaches the value of 1.0.

Abaqus provides three common mode mix formulas for computing G_{equivC}: the BK law, the power law, and the Reeder law models. The choice of model is not always clear in any given analysis; an appropriate model is best selected empirically.

BK Law

The BK law model is described in Benzeggagh (1996) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta},$$

$$G_{equiv} = G_I + G_{II} + G_{III}.$$

To define this model, you must provide G_{IC}, G_{IIC}, and η. This model provides a power law relationship combining energy release rates in Mode I, Mode II, and Mode III into a single scalar fracture criterion.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=BK

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical->Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **BK**

Power Law

The power law model is described in Wu (1965) by the following formula:

$$\frac{G_{equiv}}{G_{equivC}} = \left(\frac{G_I}{G_{IC}} \right)^{a_m} + \left(\frac{G_{II}}{G_{IIC}} \right)^{a_n} + \left(\frac{G_{III}}{G_{IIIC}} \right)^{a_o}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}, a_m, a_n$, and a_o.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=POWER

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical->Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **Power**

Reeder Law

The Reeder law model is described in Reeder (2002) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta} +$$
$$(G_{IIIC} - G_{IIC}) \left(\frac{G_{III}}{G_{II}+G_{III}} \right) \left(\frac{G_{II}+G_{III}}{G_I+G_{II}+G_{III}} \right)^{\eta},$$
$$G_{equiv} = G_I + G_{II} + G_{III}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}$, and η. The Reeder law is best applied when $G_{IIC} \neq G_{IIIC}$. When $G_{IIC} = G_{IIIC}$, the Reeder law reduces to the BK law. The Reeder law applies only to three-dimensional problems.

Input File Usage: **FRACTURE CRITERION*, TYPE=VCCT, MIXED MODE BEHAVIOR=REEDER

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical->Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **Reeder**

Defining Variable Critical Energy Release Rates

You can define a VCCT criterion with varying energy release rates by specifying the critical energy release rates at the nodes.

If you indicate that the nodal critical energy rates will be specified, any constant critical energy release rates you specify are ignored, and the critical energy release rates are interpolated from the nodes. The critical energy release rates must be defined at all nodes on the secondary surface.

Input File Usage: Use both of the following options:

**FRACTURE CRITERION*, TYPE=VCCT, NODAL ENERGY RATE
**NODAL ENERGY RATE*

Abaqus/CAE Usage: Defining variable critical energy release rates is not supported in Abaqus/CAE.

Enhanced VCCT Criterion

This criterion is available only in Abaqus/Standard.

The enhanced VCCT criterion is very similar to the original VCCT criterion described above. As in the original VCCT criterion, the fracture criterion in the general case involving Mode I, II, and III is defined as

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0.$$

The crack-tip node debonds when the fracture criterion reaches the value of 1.0. However, unlike the original VCCT criterion, you can specify two different critical fracture energy release rates: G_C for the onset of a crack and G_C^P for the growth of a crack. When the enhanced VCCT criterion is used in the general case involving

Mode I, II, and III fracture, the amount of energy dissipated associated with the release of the debonding force is controlled by the critical equivalent strain energy release rate required to propagate the crack, G^P_{equivC}, rather than by the critical equivalent strain energy release rate required to initiate the crack, G_{equivC}. The formulas for calculating G^P_{equivC} are identical to those used for G_{equivC} for different mixed-mode fracture criteria.

Input File Usage: **FRACTURE CRITERION*, TYPE=ENHANCED VCCT

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical->Fracture Criterion**, **Type**: **Enhanced VCCT**

PRX-VCCT Criterion

The Progressive Release eXplicit - Virtual Crack Closure Technique (PRX-VCCT) criterion was proposed by *Carvalho et al. (2022)* to simulate crack propagation and material delamination in a quasi-static analysis. It is available only in Abaqus/Standard.

The PRX-VCCT criterion is a modified VCCT criterion that incorporates (i) a progressive nodal release strategy within a conventional VCCT framework and (ii) an 'explicit' computation of the delamination increment based on energy release rates of the previous increment. The PRX-VCCT method has certain advantages over the conventional VCCT method, such as:

- a smoother delamination front for non-self-similar crack propagation;
- use of relatively larger element sizes without sacrificing accuracy (although the element sizes must still be small enough, as in typical fracture applications, to capture the gradients in the fields accurately); and
- better overall convergence characteristics.

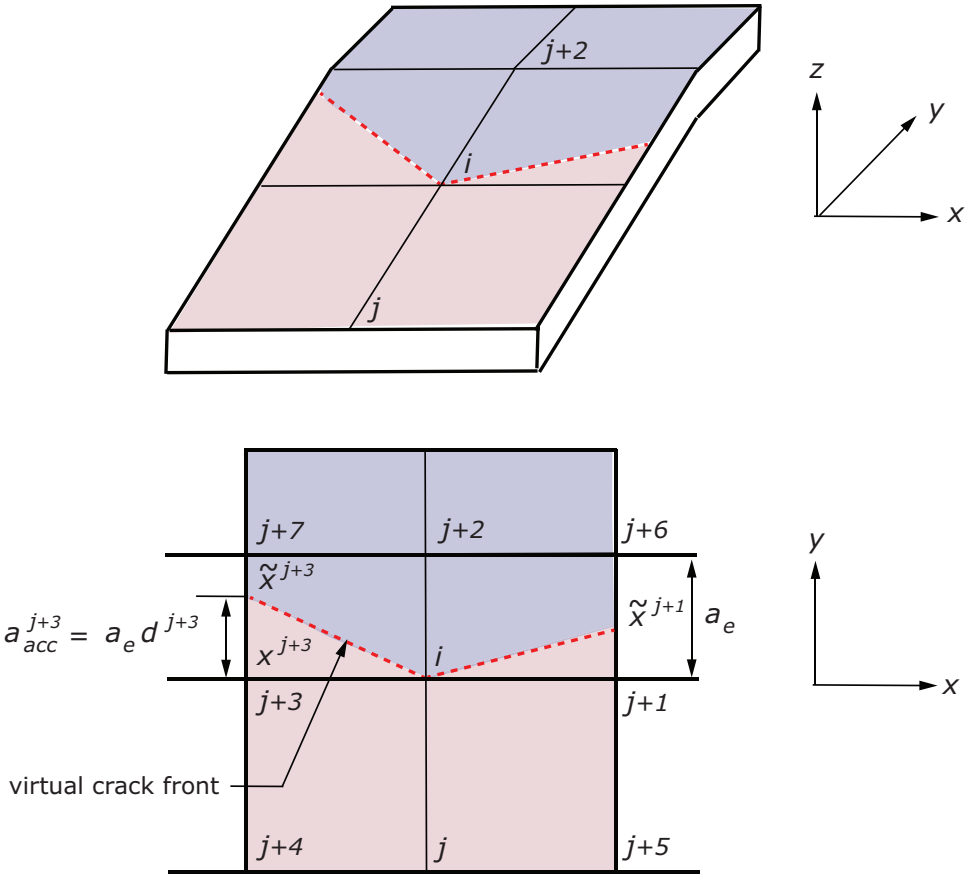


Figure 6: Delamination front area for a 3D PRX-VCCT model.

Figure 6 illustrates the bottom surface near (a segment of) an arbitrary crack front in a three-dimensional structure. In this figure nodes j, $j+4$, and $j+5$ are fully debonded; nodes $j+1$ and $j+3$ are partially bonded; and nodes i, $j+2$, $j+6$, and $j+7$ are fully bonded. The delamination front in this figure corresponds to nodes $j+1$, i, and $j+3$. The energy release rate is computed at each node along the delamination front to determine if the fracture criterion is satisfied. Therefore, in addition to computing the energy release rates at the nodes along the delamination front that are fully bonded, the PRX-VCCT method also requires the energy release rates at nodes along the delamination front that are partially debonded.

The Mode I energy release rate at node i (*Figure 6*) on the delamination front is calculated as

$$G_I = \frac{F_z^i \delta_z^j + F_z^{j+2} \delta_z^i}{2A^i},$$

where A^k, F_n^k, and δ_n^k refer to area, forces, and displacement jumps (the difference between the displacements of the top and bottom surfaces), respectively. The superscript k in a variable refers to the node at which the variable is evaluated, while the subscript z refers to the direction for the force and the displacement jumps. The above expression for G_I^i is based upon quantities at three nodes: i, j, and $j+2$, respectively, and holds irrespective of whether node i is fully or partially bonded. However, as node i transitions from a fully bonded to a partially bonded state over a number of increments, the force F_z^i reduces progressively until node i has fully debonded; at this point $F_z^i = 0.G_I^i$ is computed at node i using the expression above until it has completely debonded. The corresponding Mode II and Mode III energy release rates at node i are computed in a similar manner.

The PRX-VCCT method is fundamentally different from a traditional VCCT method, where:

- only the first term in the above expression contributes to the energy release rate calculations at node i; and
- the strain energy release rate at node i reaches a maximum once the fracture criterion is satisfied,and is not recalculated as the node transitions from a fully bonded to a partially debonded state (even though the forces at the node are progressively ramped down during debonding).

The first step in the quasi-static propagation algorithm is to evaluate the fracture criterion at each node i along the delamination front:

$$f^i = \frac{G_{equiv}}{G_{equivC}}.$$

If $f^i < 1$ for all the nodes on the delamination front, it is assumed that no delamination occurs. On the other hand, if $f^i > 1$ at any node, debonding is assumed to have occurred and all nodes along the delamination front are advanced based on the delamination growth rate equation:

$$\left(\frac{a}{\bar{t}} \right)^i = \left(f^i\right)^p,$$

where $\bar{t}$ represents a pseudo-time, a is the accumulated crack length at a node on the delamination front, and p is a ‘rate parameter’ that you specify. The default value of p is 10.

For the discussion that follows, it is useful to introduce two additional quantities:

- $a^i_{e^*}$, which represents a mesh-dependent "ligament length" ahead of the delamination front. For a regular mesh, such as the one shown in *Figure 6*, $a^i_{e^*}$ represents the distance between node i on the delamination front, and node $j+2$ ahead of it. For an arbitrary mesh, Abaqus identifies a "target node" ahead of node i in the direction of propagation of the delamination. In the latter case $a^i_{e^*}$ represents the distance between node i and the target node.
- $\lambda\,(0 < \lambda < 1)$ is a constant that you specify. It represents the maximum fraction of $a^i_{e^*}$ that the delamination is allowed to propagate in a single increment. The default value of λ is 0.1.

The delamination growth algorithm is based on a pseudo-time increment, $\Delta\bar{t}_{inc}$, which is associated with the maximum delamination increment (among all the nodes on the delamination front). To obtain $\Delta\bar{t}_{inc}$, the pseudo-time increment, $\Delta\overline{t^i}$, required to propagate the delamination by the amount $a^i_{e^*}$ at a delamination front node, i, is first determined as

$$\Delta \bar{t}^{i} = \frac{\lambda a_e^i}{\frac{da}{d\bar{t}}}.$$

The above quantity is computed at each node along the delamination front. Subsequently, $\Delta \bar{t}_{inc}$ is determined as the minimum (over all delamination front nodes) of $\Delta \bar{t}^{i}$; that is,

$$\Delta \bar{t}_{inc} = \min\left\{\Delta \bar{t}^{1}, ..., \Delta \bar{t}^{i}, ..., \Delta \bar{t}^{m}\right\}.$$

The normalized increment in delamination at a front node, i, is then computed as

$$\Delta d^{i} = \frac{\left(\frac{da}{d\bar{t}}\right)\Delta \bar{t}_{inc}}{a_{e^*}^{i}}.$$

Clearly $\Delta d^{i} = \lambda$ at the delamination front node at which $\Delta \bar{t}^{i} = \Delta \bar{t}_{inc}$, and $\Delta \bar{t}^{i} < \lambda$ at all the other nodes on the front (where $\Delta \bar{t}^{i} > \Delta \bar{t}_{inc}$). The quantity Δd^{i} is accumulated incrementally at each delamination front node. A front node is considered to have fully debonded when the accumulated value $d^{i} \geq 1$. At this point in time, the delamination front has progressed to the 'target node' ahead of the original front node i. The accumulated value of d^{i} is also used as a measure of damage to ramp down the forces at node i.

Therefore, the PRX-VCCT algorithm naturally leads to the concept of a virtual delamination front, as illustrated in *Figure 6*. As the accumulated normalized growth increment d^{i} at front node i increases from a value of 0 (totally bonded) to a value that is greater than or equal to 1.0 (totally debonded), the "virtual" delamination front moves from node i toward the target node ahead of it (node $j+2$ for the case shown in *Figure 6*). At the instance of propagation illustrated in *Figure 6*, the virtual delamination front passes through locations $\tilde{x}^{j+3}$, i, and $\tilde{x}^{j+1}$, respectively. The locations $\tilde{x}^{j+3}$ and $\tilde{x}^{j+1}$ represent delamination advancements associated with partial debonding of nodes $j+3$ and $j+1$, respectively. Full debonding at nodes $j+3$ and $j+1$ will result in the propagation of the delamination to nodes $j+7$ and $j+6$, respectively. The virtual delamination front is utilized for computing

the front tangent vectors, as well as the delamination area (used in the computation of the energy release rates) associated with each front node.

When the delamination front is at an intermediate location between nodes i and $j+2$, the force at node i can be written as

$$F_z^i = F_z^i{}_{\max}(1-d),$$

where $F_z^i{}_{\max}$ is the residual force at node i when delamination initiates.

Input File Usage: **FRACTURE CRITERION*, TYPE=PRX-VCCT, LAMBDA= , NPOWER=*p*

Abaqus/CAE Usage: The PRX-VCCT criterion is not supported in Abaqus/CAE.

Fatigue Crack Growth Criterion

This criterion is available only in Abaqus/Standard.

If you specify the fatigue crack growth criterion, you can simulate progressive delamination growth at the interfaces in laminated composites subjected to subcritical cyclic loadings. This criterion can be used only in the general fatigue crack growth approach (*Linear Elastic Fatigue Crack Growth Analysis*). The onset and delamination growth are characterized by using the Paris law, which relates the relative fracture energy release rate to crack growth rates. Different forms of the Paris law are available in Abaqus. For more information, see *Fatigue Crack Growth Using the Paris Law*.

Input File Usage: **FRACTURE CRITERION*, TYPE=FATIGUE

Abaqus/CAE Usage: The onset and growth criterion for a fatigue crack are not supported in Abaqus/CAE.

Linear Elastic Fatigue Crack Growth Analysis

The fracture energy release rates at the crack tips in the interface elements are calculated based on the above-mentioned VCCT technique.The process of delamination growth is based on a damage extrapolation technique that relies on the release of at least one element ahead of the current crack tip (once the onset criterion is satisfied) with a corresponding increase in the number of cycles. Specifically, if the criterion for the onset of delamination growth is satisfied at any crack tip node at the end of a completed cycle, N, Abaqus/Standard extends the crack length, a_N, at that node by releasing at least one element ahead of the crack tip/front at the interface. This increase in the crack length corresponds to an increase in the cycle count from the current cycle N to $N+\Delta N$, where ΔN is computed based on the Paris law. For more information on how an interfacial crack extends over an incremental number of cycles, see *Damage Extrapolation Technique at the Interface Elements*.

If $G_{max} > G_{pl}$, the interface elements at the crack tips are released by increasing the cycle number count, ΔN, by one only.

For information on how to accelerate the fatigue crack growth analysis and to provide a smooth solution for the crack front, see *Controlling Element Fracture*.

PRX-Fatigue Criterion

The Progressive Release eXplicit - Fatigue criterion was proposed by *Carvalho et al. (2022)* to simulate delamination propagation in a fatigue loading analysis. Unlike the fatigue criterion that relies on the traditional

VCCT to calculate the fracture energy release rate, the PRX-Fatigue criterion relies on the PRX-VCCT to calculate the fracture energy release rate. It is available only in Abaqus/Standard.

Similar to the PRX-VCCT method's advantages over the traditional VCCT method, the PRX-Fatigue method has certain advantages over the conventional fatigue method. For example:

- It has a smoother delamination front;
- It allows relatively larger element; and
- It has better overall convergence characteristics.

The PRX-Fatigue criterion uses the PRX-VCCT method to calculate the fracture energy release rates at the crack tips in the interface elements (see *PRX-VCCT Criterion* for more information). The PRX-Fatigue model assumes that the conditions for the onset of fatigue crack-growth are satisfied automatically. The crack growth rate is characterized by the Paris law in either of the following forms:

$$\frac{da}{dN} = c_3 G_{TMax}^{c_4} \quad \text{or} \quad \frac{da}{dN} = c_3 \Delta G^{c_4}.$$

For more information, see *Fatigue Crack Growth Using the Paris Law*.

The number of cycles needed for the delamination to propagate a distance of λa_e^i is determined using

$$\Delta N_\lambda^i = \frac{\lambda a_{e^*}^i}{\left(\frac{da}{dN}\right)^i}$$

where $a_{e^*}^i$ represents the distance between node *i* and the target node ahead of the delamination front (see *Figure 6*) and $\lambda\,(0 < \lambda < 1)$ is a constant that you specify. It represents the maximum fraction of $a_{e^*}^i$ that the delamination is allowed to propagate in an increment. The default value of λ is 0.1. Subsequently, the minimum (over all delamination front nodes) of all ΔN_λ^i is determined as

$$\Delta N_{inc} = \min\left\{\Delta N_\lambda^1, \ldots, \Delta N_\lambda^i, \ldots, \Delta N_\lambda^m\right\},$$

where the superscript *m* corresponds to the total number of nodes at the delamination front.

The normalized increment in delamination at a front node, *i,* is then computed as

$$\Delta d^i = \frac{\left(\frac{da}{dN}\right)^i \Delta N_{inc}}{a_{e^*}^i}.$$

The procedure to advance *d* is the same as that used for the *PRX-VCCT Criterion*.

Input File Usage: **FRACTURE CRITERION*, TYPE=PRX-FATIGUE, LAMBDA=

Abaqus/CAE Usage: The PRX-Fatigue criterion is not supported in Abaqus/CAE.

Specifying How a Debonding Force Is Released after a Fracture Criterion Is Met in Abaqus/Standard

After debonding, the traction between two surfaces is initially carried as equal and opposite forces at the secondary node and the corresponding point on the main surface. The debonding force is released as the crack opens and advances. Once complete debonding has occurred at a point, the bond surfaces act like standard contact surfaces with associated interface characteristics. There are several ways to release the debonding force, depending on the fracture criterion that you specify.

Specifying a Debonding Amplitude Curve

When you use the critical stress, critical crack opening displacement, or crack length versus time fracture criteria, you can define how this force is to be reduced to zero with time after debonding starts at a particular node on

the bonded surface. You specify a relative amplitude, a, as a function of time after debonding starts at a node. Thus, suppose the force transmitted between the surfaces at secondary node N is $\mathbf{T}^N\big|_0$ when that node starts to debond, which occurs at time $t^N\big|_0$. Then, for any time $t > t^N\big|_0$ the force transmitted between the surfaces at node N is $a\left(t - t^N\big|_0\right)\mathbf{T}^N\big|_0$. The relative amplitude must be 1.0 at the relative time 0.0 and must reduce to 0.0 at the last relative time point given.

The best choice of the amplitude curve depends on the material properties, specified loading, and the crack propagation criterion. If the stresses are removed too rapidly, the resulting large changes in the strains near the crack tip can cause convergence difficulties. For large-strain problems severe mesh distortion can also occur. For problems with rate-independent materials a linear amplitude curve is normally adequate. For problems with rate-dependent materials the stresses should be ramped off more slowly at the beginning of debonding to avoid convergence and mesh distortion difficulties. To reduce the likelihood of convergence and mesh distortion difficulties, you can reduce the value of the debond stress by 25% in 50% of the time to debond. The solution should not be strongly influenced by the details of the unloading procedure; if it is, this usually indicates that the mesh should be refined in the debond region.

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*
Data lines to define debonding amplitude curve

Abaqus/CAE Usage: Specifying a debonding amplitude curve is not supported in Abaqus/CAE.

Ramping Down Debonding Force for the VCCT, Enhanced VCCT, and PRX-VCCT Criteria

When the energy release rate exceeds the critical value at a crack tip, you can release the traction between the two surfaces at the crack tip in one of the following ways:

- immediately during the following increment, or
- gradually during succeeding increments with the reduction of the magnitude of the debonding force being governed by the critical fracture energy release rate.

Releasing the traction gradually is sometimes recommended to avoid sudden loss of stability when the crack tip is advanced. The table below describes which criterion can be used with each approach.

Approach	VCCT	Enhanced VCCT	PRX-VCCT
Immediate release	Supported	Not relevant; using this approach provides the same results as using the VCCT criterion	Not supported
Gradual release	Supported	Supported	Supported

Input File Usage: Use the following option to release the traction immediately:

**DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, DEBONDING FORCE=STEP

Use the following option to release the traction gradually:

**DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, DEBONDING FORCE=RAMP

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **Name:** *crack name*, **Type: Debond using VCCT**, select the step and the surface to surface (Standard) interaction, **Debonding force: Step** or **Ramp**

Ramping Down Debonding Force for the Fatigue Crack Growth Criteria

At the end of the loading cycle, the interface element that takes the fewest cycles, ΔN_{min}, to fracture is released completely with zero constraint and a zero debonding force immediately after fracture. All of the other interface elements satisfied with the accuracy of the damage extrapolation condition (as defined in *Controlling Element Fracture*) are fractured. The debonding force is removed immediately after fracture or ramped down gradually according to a scalar damage variable defined as:

$$D_{N+\Delta N} = D_N + \frac{\frac{da}{dN}\Delta N_{\min}}{a_N},$$

where $\frac{da}{dN}$ is the crack growth rate, and a_N is the element length. The debonding force becomes zero when the scalar damage variable reaches the value of 1.0 within a given tolerance, f:

$$D_{N+\Delta N} \geq 1 - f.$$

The default value of f is 0.05.

Input File Usage: Use the following options to release the traction immediately:

**SURFACE INTERACTION*, NAME=*interaction_name*
**FRACTURE CRITERION*, TYPE=FATIGUE
**
**DEBOND*, INTERACTION=*interaction_name*, DEBONDING FORCE=STEP (default)

Use the following options to release the traction gradually:

**SURFACE INTERACTION*, NAME=*interaction_name*
**FRACTURE CRITERION*, TYPE=FATIGUE, TOLERANCE=f
**
**DEBOND*, INTERACTION=*interaction_name*, DEBONDING FORCE=RAMP

Abaqus/CAE Usage: Specifying the ramping down of the debonding force for the fatigue crack growth criterion is not supported in Abaqus/CAE.

Releasing Multiple Nodes in One Increment in Abaqus/Standard

For an unstable crack growth problem, sometimes it is more efficient to allow multiple nodes at and ahead of a crack tip to debond in one increment without cutting back the increment size when the VCCT, the enhanced VCCT, or PRX-VCCT fracture criterion is satisfied. This capability is activated automatically if you specify an unstable growth tolerance, f_{tol}^u. In this case if the fracture criterion, f, is within the given unstable growth tolerance:

$$1 + f_{tol} \leq f \leq 1 + f_{tol}^u,$$

where f_{tol} is the tolerance described earlier, the time increment size by default is immediately reduced automatically to a very small value, $\beta\Delta t_{\min}$. This allows more nodes to debond until $f < 1$ for all the nodes ahead of the crack tip.

However, you can optionally specify the maximum number of cutbacks allowed, n, to be controlled by the regular tolerance, f_{tol}, prior to the activation of the unstable growth tolerance in an increment. After this limit is reached, the time increment size is automatically recovered to a larger value, $\alpha \Delta t_{pre}$, where $\Delta t_{\min}$ is the minimum time increment allowed; Δt_{pre} is the time increment size prior to the unstable crack growth; and α, β, and n are scaling parameters. The default values of α, β, and n are 0.5, 2.0, and 0, respectively.

If you do not specify a value for the unstable growth tolerance, the default value is infinity. In this case the fracture criterion, *f*, for unstable crack growth is not limited by any upper-bound value in the above equation. The forces at those debonded nodes can be completely released immediately during the following increment or ramped down gradually during succeeding increments with the reduction of the magnitude of the debonding force being governed by the critical fracture energy release rate. If the forces at the debonded nodes are ramped down gradually, an unsymmetric matrix storage and solution scheme (*Matrix Storage and Solution Scheme in Abaqus/Standard*) is used by default to improve convergence.

Input File Usage: Use one of the following sets of options:

**SURFACE INTERACTION*, NAME=*interaction_name*
**FRACTURE CRITERION*, TYPE=VCCT, UNSTABLE GROWTH
TOLERANCE=f_{tol}^{u}
**
**DEBOND*, INTERACTION=*interaction_name*, DEBONDING FORCE=RAMP

**CONTROLS*, TYPE=NO CUTBACK SCALING

or

**SURFACE INTERACTION*, NAME=*interaction_name*
**FRACTURE CRITERION*, TYPE=VCCT, UNSTABLE GROWTH
TOLERANCE=f_{tol}^{u}
**
**DEBOND*, INTERACTION=*interaction_name*, DEBONDING FORCE=STEP

Abaqus/CAE Usage: Specifying an unstable crack growth tolerance is not supported in Abaqus/CAE.

Procedures

Crack propagation analysis can be performed for static or dynamic overloadings using the following procedures:

- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Explicit Dynamic Analysis*
- *Fully Coupled Thermal-Stress Analysis*

It can also be performed for subcritical cyclic fatigue loadings using the following procedures:

- *Linear Elastic Fatigue Crack Growth Analysis*
- *Low-Cycle Fatigue Analysis Using the Direct Cyclic Approach*

Controlling Time Incrementation during Debonding in Abaqus/Standard

When automatic incrementation is used for any criteria other than VCCT, enhanced VCCT, or fatigue crack growth, you can specify the size of the time increment used just after debonding starts. By default, the time increment is equal to the last relative time specified. However, if a fracture criterion is met at the beginning of an increment, the size of the time increment used just after debonding starts will be set equal to the minimum time increment allowed in this step.

For fixed time incrementation the specified time increment value will be used as the time increment size after debonding starts if Abaqus/Standard finds it needs a smaller time increment than the fixed time increment size. The time increment size will be modified as required until debonding is complete.

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, TIME INCREMENT=*t*

Abaqus/CAE Usage: Controlling time incrementation during debonding is not supported in Abaqus/CAE.

Viscous Regularization for VCCT in Abaqus/Standard

The simulation of structures with unstable propagating cracks is challenging and difficult. Nonconvergent behavior might occur from time to time. While the usual stabilization techniques (such as contact pair stabilization and static stabilization) can be used to overcome some convergence difficulties, localized damping is included for VCCT or enhanced VCCT by using the viscous regularization technique. Viscous regularization damping causes the tangent stiffness matrix of the softening material to be positive for sufficiently small time increments.

Input File Usage: Use one of following options:

**FRACTURE CRITERION*, TYPE=VCCT, VISCOSITY=μ

**FRACTURE CRITERION*, TYPE=ENHANCED VCCT, VISCOSITY=μ

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical**->**Fracture Criterion**, **Type**: **VCCT** or **Enhanced VCCT**, **Viscosity**

Linear Scaling to Accelerate Convergence for VCCT in Abaqus/Standard

For most crack propagation simulations using VCCT or enhanced VCCT criterion, the deformation can be nearly linear up to the point of the onset of crack growth; past this point the analysis becomes very nonlinear. In this case a linear scaling method can be used to effectively reduce the solution time to reach the onset of crack growth.

Suppose that an applied "trial" load at increment $t = t_i$ is just a fraction of the critical load at the onset time of crack growth, $t = t_{crit}$. The following algorithm is used in Abaqus/Standard to quickly converge to the critical load state:

$$\Delta t_{i+1} = \left(\beta_i \sqrt{\frac{G_{equivC}}{G_{equiv}}} - 1 \right) t_i,$$

where initially β_i would be set between 0.7 and 0.9 depending on the degree of nonlinearity (the default value is 0.9). When Δt_{i+1} becomes smaller than 0.5% (indicating that the load is within 0.5% of its critical value), the next β_i is automatically set to 1.0 to cause the most critical crack-tip node to precisely reach the critical value

at the next increment. After the first crack-tip node releases, the linear scaling calculations are no longer valid and the time increment is set to the default value. Cutback is then allowed.

Input File Usage: **CONTROLS*, TYPE=VCCT LINEAR SCALING

Abaqus/CAE Usage: Step module: **Other->General Solution Controls->Edit**: *step name*, **VCCT Linear Scaling**

Nonlocal Smoothing of the Crack Tangential Directions Using Least-Squares Approximation

Crack propagation analysis is carried out on a nodal basis. The crack-tip node debonds when the fracture criterion is satisfied. The crack front is formed by connecting the individual crack-tip nodes. For coarse or unstructured meshes, the crack front might be jagged (not smooth). You can use a moving least-squares approximation by polynomials to smooth out the crack tangential directions of the crack segments along the crack front. This approximation provides a smoother pseudocrack tangential direction, leading to a more accurate evaluation of the strain energy release rates under mixed-mode loading.

Input File Usage: **FRACTURE CRITERION*, POSITION=NONLOCAL

Abaqus/CAE Usage: Nonlocal smoothing of the crack tangential directions is not supported in Abaqus/CAE.

Specifying the Approximation Used in the Least-Squares Approximation

You can use a linear, quadratic, or cubic polynomial approximation for the moving least-squares approximation to smoothen the crack tangential directions. You specify the number of terms in the polynomial.

Input File Usage: Use one of the following options:

**FRACTURE CRITERION*, NPOLY=4
**FRACTURE CRITERION*, NPOLY=7 (default)
**FRACTURE CRITERION*, NPOLY=10

Abaqus/CAE Usage: Nonlocal smoothing of the crack tangential directions is not supported in Abaqus/CAE.

Specifying the Region of the Model Used for Nonlocal Smoothing of the Crack Tangential Directions

To control the range of elements used for nonlocal smoothing of the crack tangential directions along the crack front, you can specify a radius, r_c, within which the elements around the crack tip along the crack front are included. The default radius is three times the typical element characteristic length along the crack front.

Input File Usage: **FRACTURE CRITERION*, R CRACK DIRECTION=r_c

Abaqus/CAE Usage: Nonlocal smoothing of the crack tangential directions is not supported in Abaqus/CAE.

Tips for Using the VCCT or Enhanced VCCT Criterion in Abaqus/Standard

Crack propagation problems using the VCCT or enhanced VCCT criterion are numerically challenging. The following tips will help you create a successful Abaqus/Standard model:

- An analysis with the VCCT or enhanced VCCT criterion requires small time increments. Abaqus/Standard tracks the location of the active crack front node by node when the VCCT or enhanced VCCT criterion is used. Therefore, the crack front is allowed to advance only a single node forward in any single increment (although such an advance might take place across the entire crack front in three-dimensional problems). Because an analysis using the VCCT or enhanced VCCT criterion provides detailed results of the growth of the crack, you will need small time increments, especially if the mesh is highly refined.
- Three different types of damping can be used to aid convergence for a model using the VCCT or enhanced VCCT criterion: contact stabilization, automatic or static stabilization, and viscous regularization. Contact and automatic stabilization are not specific to VCCT; they are built into Abaqus/Standard and are compatible with VCCT. Setting the value of the damping parameters is often an iterative procedure. If your VCCT model fails to converge due to unstable crack propagation, set the damping parameters to relatively high values and rerun the analysis. If the parameters are high enough, stable incrementation should return. However, the crack propagation behavior might have been modified by the damping forces and might not be physically correct. To monitor the energy absorbed by viscous damping, plot the damping energy and compare the results to the total strain energy in the model (ALLSE). When set properly, the value of the damping energy should be a small fraction of the total energy. Monitor the damping energy to ensure that the results of the VCCT simulation are reasonable in the presence of damping. When you use contact or automatic stabilization, Abaqus writes the damping energy to the variable ALLSD in the output database (`.odb`) file. When you use viscous regularization, Abaqus writes the damping energy to the variable ALLVD.
- To maximize the accuracy of the debonding simulation, try to use matched meshes between the secondary and main surfaces of the debonding contact pair.
- If you do use a mismatched mesh, you can maximize the accuracy of the simulation by using the small-sliding, surface-to-surface formulation for the contact pair (see *Contact Formulations in Abaqus/Standard*).
- Printing contact constraint information to the data (`.dat`) file allows you to review the status of the debonding contact pair at the beginning of the analysis. By printing detailed contact conditions to the message (`.msg`) file, you can track the incremental behavior of the advancing crack front during the analysis. For more information about these output requests, see *About Output*.
- You can add a small clearance to the initially unbonded portion of the debonding contact pair (*Contact Initialization for Contact Pairs in Abaqus/Standard*). The small clearance will help to eliminate unnecessary severe discontinuity iterations during incrementation as the crack begins to progress.
- Do not use tie MPCs (*General Multi-Point Constraints*) for the secondary surface in a debonding contact pair. Abaqus is unable to resolve the overconstraint presented by the MPC and the debonded contact state.
- You must have continuous main debonding surfaces.
- You might be able to help the analysis converge by adding geometric nonlinearity (even if small-sliding is used for the debonding contact pair). For more information, see *Geometric Nonlinearity*.
- For two-dimensional models with contact pairs involving higher-order underlying elements, the initially unbonded portion must extend over complete element faces. In other words, the crack tip in a two-dimensional, higher-order model must start at a corner node on the quadratic secondary surfaces. The crack tip must not start at a midside node.
- When the surface-to-surface contact formulation is used, at least two rows of elements should be used behind the crack front.

Tips for Using the PRX-VCCT Criterion in Abaqus/Standard

Usually, crack propagation models that make use of the PRX-VCCT criterion have better convergence characteristics compared to the traditional VCCT criterion. However, there are some things that you still need to consider to create a successful model in Abaqus/Standard:

- An analysis with the PRX-VCCT criterion requires relatively small time increments because of the explicit nature of the delamination algorithm. The automatic time incrementation algorithms in Abaqus/Standard might not be very effective in this situation, and you might need to specify a relatively small maximum time increment for the step in which delamination occurs. If the total time period in a static step is 1.0, then time increments of the order of $10^{-5} \leq \Delta t \leq 10^{-3}$ generally produces reasonable behavior.
- Use the UNSTABLE GROWTH TOLERANCE parameter in conjunction with the PRX-VCCT criterion only if the crack propagation is expected to be truly unstable. When this parameter is used, it is common for the specified minimum time increment to be extremely small (of the order of 10^{-10} or often significantly lower, on a time scale where the total time step is 1.0). During the unstable phase of the crack growth, the time increment reduces to extremely small values, but recovers as the crack growth becomes stable. The PRX-VCCT algorithm will work fine in such cases. However, the use of such extremely small time increments during the stable phase of crack growth could lead to oscillatory load-displacement response, and should be avoided. In summary, you should avoid the use of the UNSTABLE GROWTH TOLERANCE parameter and/or a very small minimum time increment unless the crack growth is expected to be truly unstable.
- The PRX-VCCT criterion does not require the use of the UNSYMM parameter with the step option. However, an UNSYMM parameter might still be necessary if there are other aspects of the model (for example, follower loads, or nonassociated plasticity material model) that could lead to unsymmetric stiffness contributions.
- The parameter p must be specified as an integer that is larger than *1*. The following observations regarding the dependence of analysis results on parameter p should help in choosing an appropriate value for this parameter:
  - Smaller values of p lead to a smoother crack front but relatively less accurate load-displacement response; and
  - Larger values of p lead to more accurate load-displacement response, at the expense of the smoothness of the crack front.
- A value of $\lambda \leq 0.2$ usually leads to more accurate and smoother results.

Tips for Using the VCCT Criterion in Abaqus/Explicit

Crack propagation problems using the VCCT criterion analyzed in Abaqus/Explicit benefit from the robustness of the general contact algorithm in the context of an explicit time integrator. Nevertheless, as is the case in Abaqus/Standard, these analyses remain challenging given the discontinuous nature of the fracture phenomenon. The following tips will help you create a successful Abaqus/Explicit model:

- Dynamic effects are of utmost relevance when assessing the results from a debonding analysis using the VCCT criterion. In most cases experimental and/or theoretical data are available in quasi-static settings. You must ensure that the Abaqus/Explicit analysis generates low ratios of kinetic energy to internal energy (1% or less). In practical terms this requirement often translates into avoiding the use of mass scaling in the model. Use smooth amplitudes to drive the loading to help reduce the kinetic energy in the model. Running the analysis over a longer period of time will not help in most cases because bond breakage is an inherently fast and localized process.
- If appropriate, use damping-like behavior in the materials associated with the debonding plates to reduce dynamic vibrations. Unlike Abaqus/Standard, where a pure static equilibrium is achieved at the end of a converged increment, in Abaqus/Explicit the bond breakage at a given location is associated with a dynamic overshoot beyond the static equilibrium position. If the vibrations are significant (kinetic energy is clearly

observable), the dynamic overshoot at nodes behind the crack tip might lead to premature debonding of the crack tip.

- To maximize the accuracy of the debonding simulation, use quad meshes between the secondary and main surfaces of the debonding surfaces. Avoid using elements with aspect ratios greater than 2. In most cases mesh refinement will help with obtaining a realistic result.
- Highly mismatched critical energy values between modes tend to induce crack propagation in continuously changing directions in a manner that might be unstable and unrealistic, particularly for modes II and III. Do not use such values unless experimental data suggest so.
- Use frequent field output requests to evaluate the debonding evolution as the analysis progresses. In some cases this can point to nontrivial modeling deficiencies that are difficult to identify from a simple data check analysis.
- Avoid the use of other constraints involving nodes on both surfaces of the debonding interface because the cohesive contact forces will compete with the constraint forces to achieve global equilibrium. Bond breakage might be hard to interpret in these cases.

Comparing VCCT and Cohesive Elements

Using VCCT to solve delamination problems is very similar to using cohesive elements in Abaqus. *Table 2* describes the advantages and disadvantages of the two approaches.

For an example of the use of cohesive elements, see *Delamination analysis of laminated composites*. This example also shows the effect of viscous regularization on the predicted force-displacement response.

Table 2: Comparing VCCT and cohesive elements.

VCCT	Cohesive Elements
Simulation (mechanics)-driven crack propagation along a known crack surface.	Simulation (mechanics)-driven crack propagation along a known crack surface. However, cohesive elements can also be placed between element faces as a mechanism for allowing individual elements to separate.
Models brittle fracture using LEFM only.	Model brittle or ductile fracture for LEFM or EPFM. Very general interaction modeling capability is possible.
Uses a surface-based framework. Does not require additional elements.	Require definition of the connectivity and interconnectivity of cohesive elements with the rest of the structure. For accuracy, the mesh of cohesive elements might need to be smaller than the surrounding structural mesh and the associated “cohesive zone.” As a result, cohesive elements might be more expensive.
Requires a preexisting flaw at the beginning of the crack surface. Cannot model crack initiation from a surface that is not already cracked.	Can model crack initiation from initially uncracked surfaces. The crack initiates when the cohesive traction stress exceeds a critical value.
Crack propagates when strain energy release rate exceeds fracture toughness.	Crack propagates according to cohesive damage model, usually calibrated so that the energy released when the crack is fully open equals the critical strain energy release rate.
Multiple crack fronts/surfaces can be included.	Multiple crack fronts/surfaces can be included.
In Abaqus/Standard crack surfaces are rigidly bonded when uncracked.	Crack surfaces are joined elastically when uncracked in Abaqus/Standard.
Requires user-specified fracture toughness of the bond.	Require user-specified critical traction value and fracture toughness of the bond, as well as elasticity of the bonded surface.

Measuring the Critical Strain Energy Release Properties for VCCT

You must obtain the critical strain energy release properties of the bonded surfaces for VCCT. The procedure to obtain the critical strain energy release properties is beyond the scope of this guide; however, you can refer to the following ASTM test specifications for guidance:

- ASTM D 5528-94a, “Standard Test Method for Mode I Interlaminar Fracture Toughness of Unidirectional Fiber-Reinforced Polymer Matrix Composites”
- ASTM D 6671-01, “Standard Test Method for Mixed Mode I-Mode II Interlaminar Fracture Toughness of Unidirectional Fiber-Reinforced Polymer Matrix Composites”
- ASTM D 6115-97, “Standard Test Method for Mode I Fatigue Delamination Growth Onset of Unidirectional Fiber-Reinforced Polymer Matrix Composites”

These test specifications can be found in the Annual Book of ASTM Standards, American Society for Testing and Materials, vol. 15.03, 2000.

Initial Conditions

Initial contact conditions are used to identify which part of the secondary surface is initially bonded, as explained earlier.

Boundary Conditions

Boundary conditions should not be applied to any of the nodes on the main or secondary crack surfaces, but they can be used to load the structure and cause crack propagation. Boundary conditions can be applied to any of the displacement degrees of freedom in a crack propagation analysis (*Boundary Conditions*). In a fatigue crack growth analysis, prescribed boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*).

Loads

The following types of loading can be prescribed in a crack propagation analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see *Concentrated Loads*.
- Distributed pressure forces or body forces can be applied; see *Distributed Loads*. The distributed load types available with particular elements are described in *Abaqus Elements Guide*.

For a fatigue crack growth analysis each load must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*).

Predefined Fields

The following predefined fields can be specified in a crack propagation analysis, as described in *Predefined Fields*:

- Although temperature is not a degree of freedom in stress/displacement elements, nodal temperatures can be specified as predefined fields. The specified temperature affects temperature-dependent critical stress and crack opening displacement failure criteria, if specified.
- The values of user-defined field variables can be specified. These values affect field-variable-dependent critical stress and crack opening displacement failure criteria, if specified.

The temperatures and user-defined field variables on secondary and main surfaces are averaged to determine the critical stresses and crack opening displacements.

In a fatigue crack growth analysis, the temperature values specified must be cyclic over the step: the start value must be equal to the end value (see *Amplitude Curves*). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see *Initial Conditions*). Alternatively, you can ramp the temperatures back to their initial condition values, as described in *Predefined Fields*.

Material Options

Any of the mechanical constitutive models in Abaqus/Standard can be used to model the mechanical behavior of the cracking material. See *Abaqus Materials Guide*.

Elements

Regular, rectangular meshes give the best results in crack propagation analyses. Results with nonlinear materials are more sensitive to meshing than results with small-strain linear elasticity.

First-order elements generally work best for crack propagation analysis.

Line spring elements cannot be used in crack propagation analysis.

The VCCT, enhanced VCCT, PRX-VCCT, and fatigue crack growth criteria not only support two-dimensional models (planar and axisymmetric) but also three-dimensional models with contact pairs involving first-order underlying elements (solids, shells, and continuum shells). In Abaqus/Standard use of the VCCT, enhanced VCCT, or PRX-VCCT criterion in two-dimensional models with contact pairs involving higher-order underlying elements is limited to crack fronts that are aligned with the corner nodes of the higher-order element faces. Use of the fatigue crack growth criterion with contact pairs involving higher-order underlying elements is not supported.

Output

Unless otherwise stated, the following discussions in this section are applied only to the critical stress, critical crack opening displacement, and crack length versus time criteria.

At the start of an analysis Abaqus/Standard will scan the partially bonded surfaces and identify all of the crack tips that are present in the model. The initial contact status of all the secondary surface nodes is printed in the data (`.dat`) file. At this stage Abaqus/Standard will explicitly identify all the crack tips and mark them as crack 1, crack 2, etc. The secondary and main surfaces that are associated with these cracks are also identified.

The initial contact status of all the secondary surface nodes is also printed in the data (`.dat`) file for the VCCT, enhanced VCCT, and fatigue crack growth criteria.

Printing Crack Propagation Information to the Data File

By default, crack propagation information will be printed to the data file during the analysis. For each crack that is identified Abaqus/Standard will print out the initial and current crack-tip node numbers, accumulated incremental crack length (distance from the initial crack tip to the current crack tip, measured along the secondary surface), and the current value of the user-specified fracture criterion used. Crack propagation information cannot be printed to the data file in Abaqus/Explicit.

For example, if the crack opening displacement criterion is used, the printed output during the analysis will appear as follows in the data file:

```
     CRACK TIP LOCATION AND ASSOCIATED QUANTITIES
CRACK  SECONDARY   MAIN  INITIAL  CURRENT  CUMULATIVE  CRITICAL
NUMBER SURFACE SURFACE CRACKTIP CRACKTIP INCREMENTAL COD
                       NODE #   NODE #   LENGTH
  ⋮      ⋮      ⋮        ⋮        ⋮        ⋮         ⋮
```

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **Type: Debond using VCCT**, **Write output to DAT file every** *n* **increments**

Writing Crack Propagation Information to the Results File

In Abaqus/Standard you can choose to write the crack propagation information to the results (`.fil`) file.

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, OUTPUT=FILE

Abaqus/CAE Usage: Writing crack propagation information to the results file is not supported in Abaqus/CAE.

Writing Crack Propagation Information to Both the Data File and the Results File

In Abaqus/Standard you can write the crack propagation information to both the data and the results files.

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, OUTPUT=BOTH

Abaqus/CAE Usage: Writing crack propagation information to both the data file and the results file is not supported in Abaqus/CAE.

Controlling the Output Frequency

In Abaqus/Standard you can control the output frequency in increments. By default, the crack-tip location and associated quantities will be printed every increment. Specify an output frequency of 0 to suppress crack propagation output.

Input File Usage: **DEBOND*, SECONDARY=*secondary*, MAIN=*main*, INTERACTION=*interaction_name*, FREQUENCY=*f*

Abaqus/CAE Usage: Interaction module: **Special->Crack->Create**: **Type: Debond using VCCT**, **Write output to DAT file every** *n* **increments**

Output Variables

The following bond failure quantities can be requested as surface output (see *Output to the Data and Results Files*, *Abaqus/Standard Output Variable Identifiers*, and *Abaqus/Explicit Output Variable Identifiers*) for all fracture criteria:

DBT The time when bond failure occurred. For the VCCT, enhanced VCCT, PRX-VCCT, and fatigue crack growth criteria, this is the time when debonding initiates.

DBSF Fraction of stress at bond failure that still remains.

DBS All components of remaining stress in the failed bond.

DBS1*i* 1*i* component of stress in the failed bond that remains ($i = 1, 2$).

For the VCCT, enhanced VCCT, PRX-VCCT, and fatigue crack growth criteria, the following additional variables can be also requested as surface output (see *Output to the Data and Results Files*):

CSDMG Overall value of the scalar damage variable.

BDSTAT Bond state. The bond state varies between 1.0 (fully bonded) and 0.0 (fully unbonded).

CRKLENGTH Accumulated crack length. It is measured starting from the unbonded nodes immediately behind the initial crack front. As the crack tip nodes debond, debonded element length will be added to the accumulated crack length.

OPENBC Relative displacement behind crack when the fracture criterion is met.

CRSTS All components of critical stress at failure

CRSTS1*i* 1*i* component of critical stress at failure ($i = 1, 3$).

ENRRT All components of strain energy release rate.

ENRRT1*i* 1*i* component of strain energy release rate ($i = 1, 3$).

EFENRRTR Effective energy release rate ratio, $\frac{G_{equiv}}{G_{equivC}}$.

Surface output requests provide the usual output of contact variables in addition to the above quantities. The bond failure quantities must be requested explicitly; otherwise, only the default output for contact will be given.

Abaqus/CAE provides support for the visualization of time-history plots and *X*–*Y* plots of the variables that are written to the output database.

Contour Integrals

Contour integrals can be requested for two-dimensional crack propagation analyses performed using the critical stress, critical crack opening displacement, or crack length versus time fracture criteria. If the contours are chosen so that the crack tip passes through the contour, the contour value will go to zero (as it should). Therefore, in crack propagation analysis contour integrals should be requested far enough from the crack tip that the crack tip does not pass through the contour, which is easily done by including all nodes along the bond surface in the crack-tip node set specified. See *Contour Integral Evaluation* for details on contour integral output.

Input File Template

Abaqus/Standard Analysis

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=CONTACT (, NORMAL)
Data lines to specify initial conditions
*SURFACE, NAME=secondary
Data lines to define secondary surface
*SURFACE, NAME=main
Data lines to define main surface
**
```

```
*CONTACT PAIR
secondary, main
*SURFACE INTERACTION, NAME=interaction_name
*FRACTURE CRITERION, TYPE=type, DISTANCE or NSET
Data lines to define fracture criterion
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to define material constants used in Paris law and fracture criterion
**
*STEP (, NLGEOM)
*STATIC or *VISCO or *COUPLED TEMPERATURE-DISPLACEMENT
*DEBOND, SECONDARY=secondary, MAIN=main, INTERACTION=interaction_name
Data lines to define debonding amplitude curve

*BOUNDARY
Data lines to define zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
Data lines to define loading
**
*CONTOUR INTEGRAL, CONTOURS=n, TYPE=type
**Contour integrals can be requested in a two-dimensional crack propagation analysis
*CONTACT PRINT
DBT, DBSF, DBS
*EL PRINT
JK,
*END STEP
**
*STEP
*DIRECT CYCLIC, FATIGUE
*DEBOND, SECONDARY=secondary, MAIN=main, INTERACTION=interaction_name

*BOUNDARY
Data lines to define zero-valued or nonzero cyclic boundary conditions
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
Data lines to define cyclic loading
**
*END STEP
**
```

Abaqus/Explicit Analysis

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*SURFACE, NAME=secondary
Data lines to define secondary surface
*SURFACE, NAME=main
Data lines to define main surface
**
*CONTACT CLEARANCE, NAME=clearance_name,
SEARCH NSET=initially_bonded_nodeset_name
*SURFACE INTERACTION, NAME=interaction_name
*COHESIVE BEHAVIOR
Data lines to specify elastic behavior
*FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=BK
**
*STEP
*DYNAMIC, EXPLICIT
```

```
*CONTACT
*CONTACT CLEARANCE ASSIGNMENT
Data lines to assign a clearance name to a surface pair
*CONTACT PROPERTY ASSIGNMENT
Data lines to assign a surface interaction to a surface pair
*END STEP
**
```

References

- Benzeggagh, M., and M. Kenane, “Measurement of Mixed-Mode Delamination Fracture Toughness of Unidirectional Glass/Epoxy Composites with Mixed-Mode Bending Apparatus,” *Composite Science and Technology*, vol. 56 439, 1996.
- Carvalho, N., M. Ramnath, G. Mabson, and R. Krueger, “An Explicit Delamination Propagation Algorithm to Simulate Delamination Growth under Quasi-Static and Fatigue Loading without Re-meshing Using Virtual Crack Closure Technique and Progressive Nodal Release,” *Journal of Composite Materials*, vol. 56 2063, 2022.
- Reeder, J., S. Kyongchan, P. B. Chunchu, and D. R.. Ambur, “Postbuckling and Growth of Delaminations in Composite Plates Subjected to Axial Compression” 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, vol. 1746, p. 10, 2002.
- Wu, E. M., , and R. C. Reuter Jr., “Crack Extension in Fiberglass Reinforced Plastics,” T and M Report, University of Illinois, vol. 275, 1965.

Surface-Based Fluid Modeling

In this section:

- *About Surface-Based Fluid Cavities*
- *Fluid Cavity Definition*
- *Fluid Exchange Definition*
- *Inflator Definition*

About Surface-Based Fluid Cavities

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Fluid Cavity Definition*
- *Fluid Exchange Definition*
- *Inflator Definition*

Overview

Surface-based fluid-filled cavities are modeled by:

- using standard finite elements to model the fluid-filled structure;
- using a surface definition to provide the coupling between the deformation of the fluid-filled structure and the pressure exerted by the contained fluid on the cavity boundary of the structure;
- defining the fluid behavior;
- using fluid exchange definitions to model the transfer of fluid between a cavity and the environment or between multiple cavities; and
- using inflator definitions to infuse a gas mixture into a fluid cavity to simulate the inflation of an automotive airbag.

The surface-based fluid cavity capability can be used to model a liquid or gas-filled structure. It supersedes the element-based hydrostatic fluid cavity capability in functionality and does not require the user to define fluid or fluid link elements.

Introduction

In certain applications it may be necessary to predict the mechanical response of a liquid-filled or a gas-filled structure. Examples include pressure vessels, hydraulic or pneumatic driving mechanisms, and automotive airbags. A primary difficulty in addressing such applications is the coupling between the deformation of the structure and the pressure exerted by the contained fluid on the structure. *Figure 1* illustrates a simple example of a fluid-filled structure subjected to a system of external loads. The response of the structure depends not only on the external loads but also on the pressure exerted by the fluid, which, in turn, is affected by the deformation of the structure. The surface-based fluid cavity capability provides the coupling needed to analyze situations in which the cavity can be assumed completely filled by fluid with uniform properties and state. Applications with significant spatial variation within cavities cannot be modeled with this feature. For example, consider the fluid-structure interaction and coupled Eulerian-Lagrangian capabilities for applications involving sloshing and wave propagation through a fluid (see *Eulerian Analysis* and *Fluid-Structure Interaction*).

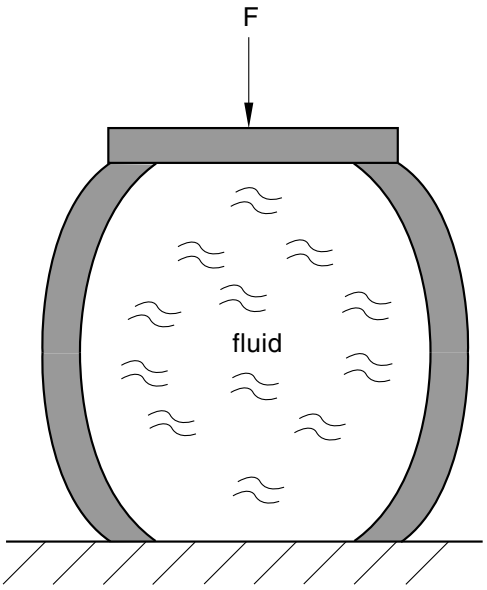


Figure 1: Fluid-filled structure.

Discretizing the Fluid Cavity

The boundary of the fluid cavity is defined by an element-based surface with normals pointing to the inside of the cavity. The underlying elements can be standard solid or structural elements as well as surface elements. Surface elements can be used to model holes in the structure or to fill in rigid regions where rigid or other load-carrying elements do not exist (see *Surface Elements*). Care must be taken when using surface elements such that nodes completely surrounded by only surface elements have proper boundary conditions.

Consider the example presented in *Figure 1*. Solid elements are defined on the top and side of the cavity as indicated in *Figure 2*. A surface element is defined on the bottom rigid boundary of the cavity where no standard elements exist. The node located at the intersection of the axis of symmetry and the lower rigid boundary of the cavity must be restrained in the r- and z-directions because it is connected only to a surface element. The surface defining the cavity is based on the underlying solid and surface elements.

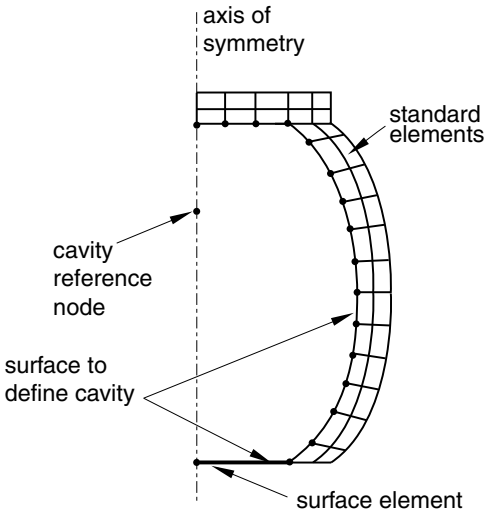


Figure 2: Axisymmetric model of fluid-filled structure.

An additional user-defined volume can be added to the actual or geometric volume of the cavity. If the boundary of the cavity is not defined by an element-based surface, the fluid cavity is assumed to have a fixed volume that is equal to the added volume.

Defining the Location of the Cavity Reference Node

A single node, known as the cavity reference node, is associated with a fluid cavity. This cavity reference node has a single degree of freedom representing the pressure inside the fluid cavity. The cavity reference node is also used in calculating the cavity volume.

If the cavity is not bounded by symmetry planes, the surface defining the cavity must completely enclose the cavity to ensure proper calculation of its volume. In this case the location of the cavity reference node is arbitrary and does not have to lie inside the cavity.

If, as a result of symmetry, only a portion of the cavity boundary is modeled with standard elements, the cavity reference node must be located on the symmetry plane or axis (*Figure 2*). If multiple symmetry planes exist, the cavity reference node must be located on the intersection of the symmetry planes (*Figure 3*). For an axisymmetric analysis the cavity reference node must be located on the axis of symmetry. These requirements are a consequence of the fluid cavity not being fully enclosed by the surface defining the cavity.

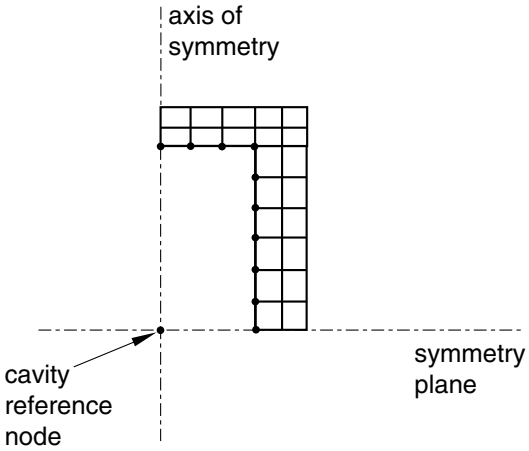


Figure 3: Axisymmetric model with additional symmetry plane.

Finite Element Calculations

The finite element calculations for surface-based cavities are performed using volume elements as described in *Hydrostatic fluid calculations*. The volume elements for a cavity are created internally by Abaqus using the surface facet geometry and the cavity reference node that you define. In Abaqus/Standard the surface facets are represented with the following element types: FAX2 and F2D2 (which are linear, 2-node, axisymmetric and planar elements, respectively) and F3D3 and F3D4 (which are linear, 3-node and 4-node three-dimensional elements, respectively). Second-order facets in Abaqus are subdivided further into multiple linear facets or elements.

Fluid Cavity Behavior

The behavior of the fluid within the fluid-filled cavity can be based either on a hydraulic or a pneumatic model. The hydraulic model can simulate nearly incompressible fluid behavior and fully incompressible behavior in Abaqus/Standard. The compressibility is introduced by defining a bulk modulus. The pneumatic model is based on an ideal gas. The gas can be defined by multiple species in Abaqus/Explicit, and you can specify the temperature of the gas or have it calculated based on the assumption of adiabatic behavior. A multi-species ideal gas with an adiabatic temperature update is an appropriate model for automotive airbags.

Modeling Flow into or out of a Cavity

There are many ways in Abaqus to model the transfer of fluid into or out of a cavity. The flow can be specified as a prescribed mass or volume flux history or can model physical mechanisms due to a pressure differential such as venting through an exhaust orifice or leakage through a porous fabric. Fluid exchange definitions are used for this purpose and can model flow between a fluid cavity and its environment or between two fluid cavities (see *Fluid Exchange Definition* for details). In addition, Abaqus/Explicit has the capability to model inflators used for the deployment of automotive airbags. Conditions at the inflator can be specified directly, or tank test data can be used (see *Inflator Definition* for details).

Modeling Multiple Chambers

Many fluid-filled systems such as airbags have multiple chambers with fluid flowing between chambers through holes or fabric leakage. In other cases it is advantageous to divide a single physical chamber into multiple chambers with fictitious walls to model a gradient in pressure across the physical chamber. Some fictitious leakage mechanisms through inter-chamber walls can be defined to obtain reasonable behavior. This can be a useful modeling technique when simulating the complex unfolding of an airbag. To model multiple chambers, define a fluid cavity for each chamber and link the fluid cavities together with the appropriate fluid exchange definitions. Averaged properties for the multi-chambered model can be output if requested (see *Fluid Cavity Definition* for details).

Defining the Fluid Inertia in a Dynamic Procedure

The inertia of the fluid inside a fluid cavity or fluid exchanged between cavities is not automatically taken into account. To add the effect of inertia, use MASS elements on the boundary of the cavity. You should make sure that the total added mass corresponds to the mass of the fluid in the cavity and that the distribution of the MASS elements is a reasonable representation of the distributed fluid mass for the type of loading to which the structure is subjected. Only the overall effect of the fluid inertia can be modeled; the uniform pressure assumption in the cavity makes it impossible to model any pressure gradient-driven fluid motions. Thus, the approach assumes that the time scale of the excitation is very long compared to typical response times for the fluid.

Modeling Contact Involving the Cavity Boundary

If a large amount of fluid is removed from a cavity or the material surrounding the cavity is very flexible, the cavity may partially collapse and portions of the cavity walls may contact each other. Self-contact of the cavity walls and contact with surrounding structures can be handled effectively by using the standard techniques available in Abaqus for modeling contact. Abaqus/Explicit can also account for the blockage of flow out of a cavity due to contacting surfaces (see *Accounting for Blockage due to Contacting Boundary Surfaces*).

Interpreting Negative Eigenvalue Messages

In some applications in Abaqus/Standard, negative eigenvalues may be encountered during the solution. These negative eigenvalues do not necessarily indicate that a bifurcation or buckling load has been exceeded. If the predicted response otherwise appears to be reasonable, these messages can be ignored. A detailed description of how negative eigenvalues can develop during the solution of hydrostatic fluid element problems is presented in *Hydrostatic fluid calculations*.

Procedures

The surface-based fluid cavity capability can be used in all procedures except coupled pore fluid diffusion/stress analysis (see *Coupled Pore Fluid Diffusion and Stress Analysis*).

Initial Conditions

The initial fluid pressure and temperature can be specified (see *Initial Conditions*). For an ideal gas the initial pressure represents the gauge pressure over and above the ambient pressure. The initial temperature should be given in the temperature scale used. Absolute zero in that temperature scale is specified separately for an ideal gas (see *Fluid Cavity Definition*).

If membrane elements are used as the underlying elements for the fluid cavity, the reference mesh (initial metric) can also be specified (see *Initial Conditions*).

Boundary Conditions

The pressure degree of freedom at the cavity reference node (degree of freedom number 8) is a primary variable in the problem. Thus, it can be prescribed by defining a boundary condition (see *Boundary Conditions*), similar to the way displacements of structural nodes can be prescribed. Prescribing the pressure at the cavity reference node is equivalent to applying a uniform pressure to the cavity boundary using a distributed load definition (see *Distributed Loads*).

If the pressure is prescribed with a boundary condition, the fluid volume is adjusted automatically to fill the cavity (that is, fluid is assumed to enter and leave the cavity as needed to maintain the prescribed pressure). This behavior is useful in situations where a cavity is deformed prior to the introduction of the effect of the fluid. In a subsequent step you can remove the boundary condition on the pressure degree of freedom (see *Removing Boundary Conditions*), thus "sealing" the cavity with the current fluid volume.

Loads

Distributed pressures and body forces, as well as concentrated nodal forces, can be applied to the fluid-filled structure, as described in *Concentrated Loads* and *Distributed Loads*.

Predefined Fields

Predefined temperature fields and user-defined field variables can be defined for both fluid-filled structures and the enclosed fluids, as described in *Predefined Fields*.

Temperatures

Fluid temperatures can be specified at all cavity reference nodes as predefined fields (see *Predefined Temperature*), unless an adiabatic process is specified or a coupled temperature-displacement procedure is used. Any difference between the applied and initial temperatures will cause thermal expansion for a pneumatic fluid and for a hydraulic fluid if a thermal expansion coefficient is given. A specified temperature field can also affect temperature-dependent material properties, if any exist, for both fluid-filled structures and enclosed fluids.

Field Variables

The values of user-defined field variables can be specified at all cavity reference nodes (see *Predefined Field Variables*). These values will affect field-variable-dependent material properties for the enclosed fluid.

Output

The state of the fluid inside the cavity is available for history output using the nodal output variables PCAV and CVOL, which represent the gauge fluid pressure and cavity volume, respectively. In steady-state dynamic procedures the magnitude and phase angle of the fluid pressure can be obtained as nodal variable PPOR.

Abaqus/Explicit also provides output for the cavity temperature, cavity surface area, and mass of the fluid (nodal output variables CTEMP, CSAREA, and CMASS, respectively). Output variable CTEMP is available only when an ideal gas model is used under adiabatic conditions. If the node set for which the output request is made contains

more than one fluid cavity, the time histories of the average fluid pressure, total volume, average fluid temperature, sum of all the external cavity surface areas, and total mass of these cavities will also be output by using the nodal output variables APCAV, TCVOL, ACTEMP, TCSAREA, and TCMASS, respectively.

In Abaqus/Explicit, when the model includes fluid exchange definitions, use nodal output variables CMFL and CMFLT to obtain history output of the total mass flow rate and total accumulated mass flow out of a cavity and CEFL and CEFLT to obtain history output of the total heat energy flow rate and total accumulated heat energy flow out of a cavity. If more than one fluid exchange is defined for a cavity, time histories of the mass or heat energy flow rate and accumulated mass or heat energy flow out of the cavity for each fluid exchange will also be output.

If the fluid cavity is modeled by a mixture of ideal gases, time histories of the molecular mass fraction of each fluid species inside the fluid cavity can be obtained by using nodal output variable CMF.

If inflators are used, use nodal output variables MINFL, MINFLT, and TINFL to obtain time histories of mass flow rate, accumulated mass flow, and inflator temperature for each inflator definition (see *Abaqus/Explicit Output Variable Identifiers*).

Limitations

In the context of modeling a fluid cavity in Abaqus/Standard or Abaqus/Explicit, temperature can be considered as a field variable and is not a degree of freedom of the system. Therefore, thermal energy is not included in the energy output.

Input File Template

An Analysis with Hydrostatic Fluid:

```
*HEADING
…
*FLUID CAVITY, NAME=cavity_name, BEHAVIOR=behavior_name,
REF NODE=cavity_reference_node, SURFACE=surface_name
*FLUID BEHAVIOR, NAME=behavior_name
*FLUID DENSITY
Data line to define density
*FLUID BULK MODULUS
Data line to define  bulk modulus
*FLUID EXPANSION
Data line to define thermal expansion
**
*FLUID EXCHANGE, NAME=exchange_name, PROPERTY=exchange_property_name
cavity_reference_node
*FLUID EXCHANGE PROPERTY, NAME=exchange_property_name, TYPE=MASS FLUX
Data line to define mass flow rate per unit area
**
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data line to define initial temperature
*INITIAL CONDITIONS, TYPE=FLUID PRESSURE
Data line to define initial pressure
**
*STEP
**
*TEMPERATURE
Data line to define temperature
*FLUID EXCHANGE ACTIVATION
exchange_name
**
*END STEP
```

An Airbag Analysis with a Mixture of Ideal Gases:

```
*HEADING
…
*FLUID CAVITY, NAME=chamber_1, MIXTURE=MOLAR FRACTION, ADIABATIC,
REF NODE=chamber_1_reference_node, SURFACE=surface_name_1
blank line
Oxygen, 0.2
Nitrogen, 0.75
Carbon_dioxide, 0.05
**
*FLUID CAVITY, NAME=chamber_2, BEHAVIOR=Air, ADIABATIC,
REF NODE=chamber_2_reference_node, SURFACE=surface_name_2
blank line
**
*FLUID BEHAVIOR, NAME=Air
*CAPACITY, TYPE=POLYNOMIAL
Data line to define heat capacity coefficient
*MOLECULAR WEIGHT
Data line to define molecular weight
**
*FLUID BEHAVIOR, NAME=Oxygen
*CAPACITY, TYPE=POLYNOMIAL
Data line to define heat capacity coefficient
*MOLECULAR WEIGHT
Data line to define molecular weight
**
*FLUID BEHAVIOR, NAME=Nitrogen
*CAPACITY, TYPE=POLYNOMIAL
Data line to define heat capacity coefficient
*MOLECULAR WEIGHT
Data line to define molecular weight
**
*FLUID BEHAVIOR, NAME=Carbon_dioxide
*CAPACITY, TYPE=POLYNOMIAL
Data line to define heat capacity coefficient
*MOLECULAR WEIGHT
Data line to define molecular weight
**
*FLUID INFLATOR, NAME=inflator, PROPERTY=inflator_property
chamber_1_reference_node
*FLUID INFLATOR PROPERTY, NAME=inflator_property,
TYPE=TEMPERATURE AND MASS
Data lines to define mass flow rate and gas temperature
*FLUID INFLATOR MIXTURE, TYPE=MOLAR FRACTION, NUMBER SPECIES=2
Carbon_dioxide, Nitrogen
Table to define molecular mass fraction
**
*FLUID EXCHANGE, NAME=exhaust, PROPERTY=exhaust_behavior
chamber_1_reference_node
*FLUID EXCHANGE PROPERTY, NAME=exhaust_behavior, TYPE=ORIFICE
Data line to specify orifice behavior
*FLUID EXCHANGE, NAME=leakage_1, PROPERTY=fabric_behavior
chamber_1_reference_node
*FLUID EXCHANGE, NAME=leakage_2, PROPERTY=fabric_behavior
chamber_2_reference_node
*FLUID EXCHANGE PROPERTY, NAME=fabric_behavior, TYPE=FABRIC LEAKAGE
```

```
Data line to specify fabric leakage behavior
**
*FLUID EXCHANGE, NAME=chamber_wall, PROPERTY=wall_behavior,
EFFECTIVE AREA=
chamber_1_reference_node, chamber_2_reference_node
*FLUID EXCHANGE PROPERTY, NAME=wall_behavior, TYPE=ORIFICE
Data line to specify orifice behavior
**
*AMPLITUDE, NAME=amplitude_name
Data line to define amplitude variations
*PHYSICAL CONSTANTS, UNIVERSAL GAS CONSTANT=
**
*INITIAL CONDITIONS, TYPE=FLUID PRESSURE
Data line to define initial pressure
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data line to define initial temperature
**
*STEP
**
*FLUID EXCHANGE ACTIVATION
exhaust, leakage_1, leakage_2,  chamber_wall
*FLUID INFLATOR ACTIVATION, INFLATION TIME AMPLITUDE=amplitude_name
inflator
**
*END STEP
```

Fluid Cavity Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surface-Based Fluid Cavities*
- *Fluid Exchange Definition*
- **CAPACITY*
- **FLUID BEHAVIOR*
- **FLUID BULK MODULUS*
- **FLUID CAVITY*
- **FLUID DENSITY*
- **MOLECULAR WEIGHT*
- *Defining a fluid cavity interaction*
- *Defining a fluid cavity interaction property*

Overview

A surface-based fluid cavity:

- can be used to model a liquid-filled or gas-filled structure;
- is associated with a node known as the cavity reference node;
- is defined by specifying a surface that fully encloses the cavity;
- is applicable only for situations where the pressure and temperature of the fluid in a particular cavity are uniform at any point in time;
- can be used to model an airbag using the assumptions of an ideal gas mixture under adiabatic conditions; and
- has a name that can be used to identify history output associated with the cavity.

Defining the Fluid Cavity

You must associate a name with each fluid cavity.

Input File Usage: **FLUID CAVITY*, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**, **Name**: *name*

Specifying the Cavity Reference Node

Every fluid cavity must have an associated cavity reference node. Along with the fluid cavity name, the reference node is used to identify the fluid cavity. In addition, it may be referenced by fluid exchange and inflator definitions. The reference node should not be connected to any elements in the model.

Input File Usage: **FLUID CAVITY*, REF NODE=*n*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: select the fluid cavity reference node

Specifying the Boundary of the Fluid Cavity

The fluid cavity must be completely enclosed by finite elements unless symmetry planes are modeled (see *About Surface-Based Fluid Cavities*). Surface elements can be used for portions of the cavity surface that are not structural. The boundary of the cavity is specified using an element-based surface covering the elements that surround the cavity with surface normals pointing inward. By default, an error message is issued if the underlying elements of the surface do not have consistent normals. Alternatively, you can skip the consistency checking for the surface normals.

Input File Usage: Use the following option to define the surface with consistent normal checking:

**FLUID CAVITY*, SURFACE=*surface_name*, CHECK NORMALS=YES

Use the following option to define the surface without consistent normal checking:

**FLUID CAVITY*, SURFACE=*surface_name*, CHECK NORMALS=NO

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: select the fluid cavity boundary surface; toggle on or off **Check surface normals**

Specifying Additional Volume in a Fluid Cavity

An additional volume can be specified for a fluid cavity. The additional volume will be added to the actual volume when the boundary of the cavity is defined by a specified surface. If you do not specify a surface forming the boundary of the fluid cavity, the fluid cavity is assumed to have a fixed volume that is equal to the added volume. In Abaqus/Standard, along with the added volume, the surface forming the boundary of the fluid cavity must be specified.

Input File Usage: **FLUID CAVITY*, ADDED VOLUME=*r*

Abaqus/CAE Usage: Specification of additional volume is not supported in Abaqus/CAE.

Specifying the Minimum Volume

When the volume of a fluid cavity is extremely small, transients in an explicit dynamic procedure can cause the volume to go to zero or even negative causing the effective cavity stiffness values to tend to infinity. To avoid numerical problems, you can specify a minimum volume for the fluid in Abaqus/Explicit. If the volume of the cavity (which is equal to the actual volume plus the added volume) drops below the minimum, the minimum value is used to evaluate the fluid pressure.

You can specify the minimum volume either directly or as the initial volume of the fluid cavity. If the latter method is used and the initial volume of the fluid cavity is a negative value, the minimum volume is set equal to zero.

Input File Usage: Use the following option to specify the minimum volume directly:

**FLUID CAVITY*, MINIMUM VOLUME=*minimum volume*

Use the following option to specify the minimum volume to be equal to the initial volume:

**FLUID CAVITY*, MINIMUM VOLUME=INITIAL VOLUME

Abaqus/CAE Usage: Specification of a minimum volume is not supported in Abaqus/CAE.

Defining the Fluid Cavity Behavior

The fluid cavity behavior governs the relationship between cavity pressure, volume, and temperature. A fluid cavity in Abaqus/Standard can contain only a single fluid. In Abaqus/Explicit a cavity can contain a single fluid or a mixture of ideal gases.

Fluid Behavior with a Homogeneous Fluid

To define a fluid cavity behavior made of a single fluid, specify a single fluid behavior to define the fluid properties. You must associate the fluid behavior with a name. This name can then be used to associate a certain behavior with a fluid cavity definition.

Input File Usage: Use the following options:

**FLUID CAVITY*, NAME=*fluid_cavity_name*, BEHAVIOR=*behavior_name*
**FLUID BEHAVIOR*, NAME=*behavior_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**, **Name**: *behavior_name*

Fluid Behavior with a Mixture of Ideal Gases in Abaqus/Explicit

In Abaqus/Explicit you can define a fluid cavity behavior made of multiple gas species. To define a fluid cavity behavior made of multiple gas species, you specify multiple fluid behaviors to define the fluid properties. Specify the names of the fluid behaviors and the initial mass or molar fractions defining the mixture to associate a certain group of behaviors with a fluid cavity definition.

Input File Usage: Use the following options to define the fluid cavity mixture in terms of the initial mass fraction:

**FLUID BEHAVIOR*, NAME=*behavior_name*
**FLUID CAVITY*, NAME=*fluid_cavity_name*, MIXTURE=MASS FRACTION
out-of-plane surface thickness (if required; otherwise, blank)
behavior_name, *initial mass fraction*
...

Use the following options to define the fluid cavity mixture in terms of the initial molar fraction:

**FLUID BEHAVIOR*, NAME=*behavior_name*
**FLUID CAVITY*, NAME=*fluid_cavity_name*, MIXTURE=MOLAR FRACTION
out-of-plane surface thickness (if required; otherwise, blank)
behavior_name, *initial molar fraction*
...

Abaqus/CAE Usage: Specification of ideal gas mixtures is not supported in Abaqus/CAE.

User-Defined Fluid Behavior in Abaqus/Standard

In Abaqus/Standard the fluid behavior can be defined in user subroutine *UFLUID*.

Input File Usage: **FLUID BEHAVIOR*, USER

Abaqus/CAE Usage: User subroutine *UFLUID* is not supported in Abaqus/CAE.

Defining the Ambient Pressure for a Fluid Cavity

For pneumatic fluids the equilibrium problem is generally expressed in terms of the "gauge" pressure in the fluid cavity (that is, ambient atmospheric pressure does not contribute to the loading of the solid and structural parts of the system). You can choose to convert gauge pressure to absolute pressure $\tilde{p}$ as used in the constitutive law. For hydraulic fluids you can define the ambient pressure, which can be used to calculate the pressure difference in the fluid exchange between a fluid cavity and its environment. The pressure value given as degree of freedom 8 at the cavity reference node is the value of the gauge pressure. The ambient pressure, p_A, is assumed to be zero if you do not specify it.

Input File Usage: **FLUID CAVITY*, AMBIENT PRESSURE=p_A

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: toggle on **Specify ambient pressure**: p_A

Isothermal Process

For hydraulic fluids and pneumatic fluids in problems of long time duration, it is reasonable to assume that the temperature is constant or a known function of the environment surrounding the cavity. In this case the temperature of the fluid can be defined by specifying initial conditions (see *Defining Initial Temperatures*) and predefined temperature fields (see *Predefined Temperature*) at the cavity reference node. For a pneumatic fluid the pressure and density of the gas are calculated from the ideal gas law, conservation of mass, and the predefined temperature field.

Defining the Ambient Temperature for a Fluid Cavity

For pneumatic fluids with adiabatic behavior the ambient temperature is needed when the heat energy flow is defined between a single cavity and its environment and the flow definition is based on analysis conditions. The ambient temperature, θ_A, is assumed to be zero if you do not specify it.

Input File Usage: **FLUID CAVITY*, AMBIENT TEMPERATURE=θ_A

Abaqus/CAE Usage: Specification of ambient temperature is not supported in Abaqus/CAE.

Hydraulic Fluids

The hydraulic fluid model is used to model nearly incompressible fluid behavior and fully incompressible fluid behavior in Abaqus/Standard. Compressibility is introduced by assuming a linear pressure-volume relationship. The required parameters for compressible behavior are the bulk modulus and the reference density. You omit the bulk modulus to specify fully incompressible behavior in Abaqus/Standard. Specifying a high bulk modulus

may affect the stable time increment in Abaqus/Explicit. Temperature dependence of the density can be modeled as a thermal expansion of the fluid.

Input File Usage: **FLUID CAVITY*, BEHAVIOR=*behavior_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**

Defining the Reference Fluid Density

The reference fluid density, ρ_R, is specified at zero pressure and the initial temperature, θ_I:

$$\rho_R = \rho(0, \theta_I).$$

Input File Usage: **FLUID DENSITY*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid density**: *density*

Defining the Fluid Bulk Modulus for Compressibility

The compressibility is described by the bulk modulus of the fluid:

$$p = -K\left(\frac{V(p,\theta)-V_0(\theta)}{V_0(\theta_I)}\right) = -K\rho_R\left(\rho^{-1}(p,\theta) - \rho_0^{-1}(\theta)\right),$$

where

p

is the current pressure,

θ

is the current temperature,

K

is the fluid bulk modulus,

$V(p,\theta)$

is the current fluid volume,

$\rho(p,\theta)$

is the density at current pressure and temperature,

$V_0(\theta)$

is the fluid volume at zero pressure and current temperature,

$V_0(\theta_I)$

is the fluid volume at zero pressure and initial temperature, and

$\rho_0(\theta)$

is the density at zero pressure and current temperature.

It is assumed that the bulk modulus is independent of the change in fluid density. However, the bulk modulus can be specified as a function of temperature or predefined field variables.

Input File Usage: **FLUID BULK MODULUS*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid Bulk Modulus** tabbed page: toggle on **Specify fluid bulk modulus**, and enter the modulus value in the table

Use the following options to include temperature and field variable dependence:

Toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Defining the Fluid Expansion

The thermal expansion coefficients are interpreted as total expansion coefficients from a reference temperature, which can be specified as a function of temperature or predefined field variables. The change in fluid volume due to thermal expansion is determined as follows:

$$V_0(\theta) = V_0(\theta_I)\left[1 + 3\alpha(\theta)(\theta - \theta_0) - 3\alpha(\theta_I)(\theta_I - \theta_0)\right],$$

where θ_0 is the reference temperature for the coefficient of thermal expansion and $\alpha(\theta)$ is the mean (secant) coefficient of thermal expansion.

If the coefficient of thermal expansion is not a function of temperature or field variables, the value of θ_0 is not needed.

Thermal expansion can also be expressed in terms of the fluid density:

$$\rho_0(\theta) = \rho_R / \left[1 + 3\alpha(\theta)(\theta - \theta_0) - 3\alpha(\theta_I)(\theta_I - \theta_0)\right].$$

Input File Usage: **FLUID EXPANSION*, ZERO=θ_0

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid Expansion** tabbed page: toggle on **Specify fluid thermal expansion coefficients**, and enter the mean coefficient of thermal expansion in the table

Use the following options to include temperature and field variable dependence:

Toggle on **Use temperature-dependent data**, **Reference temperature**: θ_0, **Number of field variables**: *n*

Pneumatic Fluids

Compressible or pneumatic fluids are modeled as an ideal gas (see *Equation of State*). The equation of state for an ideal gas (or the ideal gas law) is given as

$$\tilde{p} = \rho R \left(\theta - \theta^Z\right),$$

where the absolute (or total) pressure $\tilde{p}$ is defined as

$$\tilde{p} = p + p_A,$$

and p_A is the ambient pressure, p is the gauge pressure, R is the gas constant, θ is the current temperature, and θ^Z is absolute zero on the temperature scale being used. The gas constant, R, can also be determined from the universal gas constant, $\tilde{R}$, and the molecular weight, MW, as follows:

$$R = \frac{\tilde{R}}{MW}.$$

Conservation of mass gives the change of mass in the fluid cavity as

$$\dot{m} = \dot{m}_{in} - \dot{m}_{out},$$

where m is the mass of the fluid, $\dot{m}_{in}$ is the mass flow rate into the fluid cavity, and $\dot{m}_{out}$ is the mass flow rate out of the fluid cavity.

Defining the Molecular Weight

You must specify the value of the molecular weight of the ideal gas, MW.

Input File Usage: **MOLECULAR WEIGHT*
MW

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**, **Ideal gas molecular weight**: MW

Specifying the Value of the Universal Gas Constant

You can specify the value of the universal gas constant, $\tilde{R}$.

Input File Usage: **PHYSICAL CONSTANTS*, UNIVERSAL GAS CONSTANT=$\tilde{R}$

Abaqus/CAE Usage: All modules: **Model->Edit attributes->*model name***: **Physical Constants**: toggle on **Universal gas constant**: $\tilde{R}$

Specifying the Value of Absolute Zero

You can specify the value of absolute zero temperature, θ^Z.

Input File Usage: **PHYSICAL CONSTANTS*, ABSOLUTE ZERO=θ^Z

Abaqus/CAE Usage: All modules: **Model->Edit attributes->*model name***: **Physical Constants**: toggle on **Absolute zero temperature**: θ^Z

Adiabatic Process

By default, the fluid temperature is defined by the predefined temperature field at the cavity reference node. However, for rapid events the fluid temperature in Abaqus/Explicit can be determined from the conservation of energy assumed in an adiabatic process. With this assumption, no heat is added or removed from the cavity except by transport through fluid exchange definitions or inflators. An adiabatic process is typically well suited for modeling the deployment of an airbag. During deployment, the gas jets out of the inflator at high pressure and cools as it expands at atmospheric pressure. The expansion is so quick that no significant amount of heat can diffuse out of the cavity.

The energy equation can be obtained from the first law of thermodynamics. By neglecting the kinetic and potential energy, the energy equation for a fluid cavity is given by

$$\frac{d(mE)}{dt} = \dot{m}_{in} H_{in} - \dot{m}_{out} H_{out} - \dot{W} - \dot{Q},$$

where the work done by the fluid cavity expansion is given as

$$\dot{W} = p\dot{V}$$

and $\dot{Q}$ is the heat energy flow rate due to the heat transfer through the surface of the fluid cavity. A positive value for $\dot{Q}$ will generate the heat energy flow out of the primary fluid cavity. The specific energy is given by

$$E = E_I + \int_{\theta_I}^{\theta} c_v\,(T)\,dT,$$

where E_I is the initial specific energy at the initial temperature θ_I, c_v is the specific heat at constant volume (or the constant volume heat capacity), which depends only upon temperature for an ideal gas, H is the specific enthalpy, and V is the volume occupied by the gas. By definition, the specific enthalpy is

$$H = H_I + \int_{\theta_I}^{\theta} c_p\,(T)\,dT,$$

where H_I is the initial specific enthalpy at the initial (or reference) temperature θ_I and c_p is the specific heat at constant pressure, which depends only upon temperature for an ideal gas. The pressure, temperature, and density of the gas are obtained by solving the ideal gas law, the energy balance, and mass conservation.

Adiabatic behavior will always be used for the fluid cavity if an adiabatic or coupled procedure is used.

Input File Usage: **FLUID CAVITY*, ADIABATIC

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: **Property definition**: **Pneumatic**, toggle on **Use adiabatic behavior**

Defining the Heat Capacity at Constant Pressure

You must define the heat capacity at constant pressure for the ideal gas. It can be defined either in polynomial or tabular form. The polynomial form is based on the Shomate equation according to the National Institute of Standards and Technology. The constant pressure molar heat capacity can be expressed as

$$\tilde{c}_p = \tilde{a} + \tilde{b}\left(\theta - \theta^Z\right) + \tilde{c}\left(\theta - \theta^Z\right)^2 + \tilde{d}\left(\theta - \theta^Z\right)^3 + \frac{\tilde{e}}{\left(\theta - \theta^Z\right)^2},$$

where the coefficients $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, and $\tilde{e}$ are gas constants. These gas constants together with molecular weight are listed in *Table 1* for some gases that are often used in airbag simulations. The constant pressure heat capacity can then be obtained by

$$c_p = \frac{\tilde{c}_p}{MW}.$$

The constant volume heat capacity, c_v, can be determined by

$$c_v = c_p - R.$$

Table 1: Properties of some commonly used gases (SI units).

Gas	MW	$\tilde{a}$	$\tilde{b}$	$\tilde{c}$	$\tilde{d}$	$\tilde{e}$	θ
			(× 10^{-3})	(× 10^{-6})	(× 10^{-9})	(× 10^{6})	(kelvin)
Air	0.0289	28.110	1.967	4.802	–1.966	0.0	273–1800
Nitrogen	0.028	26.092	8.218	–1.976	0.1592	0.0444	298–6000
Oxygen	0.032	29.659	6.137	–1.186	0.0957	–0.219	298–6000
Hydrogen	0.00202	33.066	–11.36	11.432	–2.772	–0.158	273–1000
Carbon monoxide	0.028	25.567	6.096	4.054	–2.671	0.131	298–1300
Carbon dioxide	0.044	24.997	55.186	–33.691	7.948	–0.136	298–1200
Water vapor	0.0180	32.240	1.923	0.105	–3.595	0.0	273–1800

You can use the polynomial form for specifying the heat capacity at constant pressure, in which case you enter the coefficients $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, and $\tilde{e}$. Alternatively, you can define a table of constant pressure heat capacity versus temperature and any predefined field variables.

Input File Usage: Use the following option to specify the heat capacity in polynomial form:

**CAPACITY*, TYPE=POLYNOMIAL
$\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, $\tilde{e}$

Use the following option to specify the heat capacity in tabular form:

**CAPACITY*, TYPE=TABULAR, DEPENDENCIES=*n*

$\tilde{c}_p$, *temperature, field_variable_1, etc...*
...

Abaqus/CAE Usage: Use the following option to specify the heat capacity in polynomial form:

Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**, toggle on **Specify molar heat capacity**: **Polynomial**, **Polynomial Coefficients**: $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, $\tilde{e}$

Use the following option to specify the heat capacity in tabular form:

Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**: toggle on **Specify molar heat capacity**: **Tabular**: enter the molar heat capacity

Use the following options to include temperature and field variable dependence in the table:

Toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

A Mixture of Ideal Gases

Abaqus/Explicit can model a mixture of ideal gases in the fluid cavity. For ideal gas mixtures the Amagat-Leduc rule of partial volumes is used to obtain an estimate of the molar-averaged thermal properties (Van Wylen and Sonntag, 1985). Let each species have constant pressure and volume heat capacities, c_{p_i} and c_{v_i}; molecular weight, MW_i; and mass fraction, f_i. The constant pressure and volume heat capacities for the mixed gas are then given by

$$c_p = \sum_i f_i c_{p_i},$$

$$c_v = \sum_i f_i c_{v_i},$$

and the molecular weight is given by

$$MW = 1/\sum_i \frac{f_i}{MW_i}.$$

The specific energy and enthalpy for the mixed gas are then given by

$$E = \sum_i f_i E_i,$$

$$H = \sum_i f_i H_i.$$

The energy flow entering the fluid cavity is given by

$$\dot{m}_{in} H_{in} = \sum_i \dot{m}_{in_i} H_{in_i},$$

and the energy flow out of the fluid cavity is given by

$$\dot{m}_{out} H_{out} = \sum_i \dot{m}_{out_i} H_{out_i}.$$

Using the properties of a mixture of ideal gases as given above, the pressure and temperature can be obtained from the ideal gas law and the energy equation.

Averaged Properties for Multiple Fluid Cavities

If the output of the state of the fluid inside the cavity is requested for a node set that contains more than one fluid cavity, the averaged properties of the multiple fluid cavities will also be output automatically. The average pressure is calculated by volume weighting cavity pressure contributions. The average temperature for an adiabatic ideal gas is obtained by mass weighting cavity temperature contributions. Let each fluid cavity have pressure p_k, temperature θ_k, volume V_k, gas constant R_k, and mass m_k. The average pressure of the fluid cavity cluster is then defined as

$$p_{avg} = \sum_k p_k V_k / \sum_k V_k,$$

and the average temperature is

$$\theta_{avg} = \sum_k R_k \theta_k m_k / \sum_k R_k m_k.$$

References

- Van Wylen, G. J., and R. E. Sonntag, Fundamentals of Classical Thermodynamics, Wiley, New York, 1985.

Fluid Exchange Definition

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Surface-Based Fluid Cavities*
- *Fluid Cavity Definition*
- **FLUID EXCHANGE*
- **FLUID EXCHANGE PROPERTY*
- **FLUID EXCHANGE ACTIVATION*
- *VUFLUIDEXCH*
- *VUFLUIDEXCHEFFAREA*
- *Defining a fluid exchange interaction*
- *Defining a fluid exchange interaction property*

Overview

A fluid exchange definition:

- can be used to model flow between a single fluid cavity and its environment or flow between two fluid cavities;
- can be used to prescribe mass- or volume-based flux into or out of a cavity;
- can model the venting of a cavity through an exhaust orifice;
- can model flow through cavity walls such as leakage through a porous fabric;
- can be used to prescribe heat loss through a cavity surface due to heat transfer;
- can take the local material state into account;
- can account for blockage due to contacting boundary surfaces; and
- has a name that can be used to identify history output of mass flow rates out of a cavity.

Defining Fluid Exchange

The fluid exchange capability is very general and can be used to define flow in and out of a cavity either as a prescribed function or based on the pressure difference arising from analysis conditions. The flow behavior in Abaqus/Standard is based on mass fluid flow, and the behavior in Abaqus/Explicit can be based on mass fluid flow or heat energy flow. You must associate the fluid exchange definition with a name.

Input File Usage: **FLUID EXCHANGE*, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**, **Name**: *name*

Flow between a Single Cavity and Its Environment

To define flow between a fluid cavity and its environment in Abaqus/Explicit, specify the single reference node associated with the fluid cavity. In the discussion that follows this fluid cavity is referred to as the primary cavity. When the flow is defined as a prescribed function, the flow can either be into or out of the primary cavity. If the flow is into the cavity, the properties of the material flowing in are assumed to be the instantaneous properties

of the material in the cavity itself. When the flow behavior is based on analysis conditions, the mass flow can occur only out of the primary cavity but the heat energy flow can be either into or out of the primary cavity. For the case of mass flow Abaqus will use the fluid cavity pressure and the specified constant ambient pressure to calculate the pressure difference used to determine the mass flow rate. For the case of heat energy flow Abaqus/Explicit will use the fluid cavity temperature and the specified constant ambient temperature to calculate the temperature difference used to determine the heat energy flow rate.

Input File Usage: Use the following options:

**FLUID CAVITY*, NAME=*primary_cavity_name*,
REF NODE=*primary_cavity_reference_node*
**FLUID EXCHANGE*, NAME=*fluid_exchange_name*
primary_cavity_reference_node

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Definition**: **To environment**, **Fluid cavity interaction**: *name*, **Fluid exchange property**: *name*

Flow between Two Fluid Cavities

To define flow between two fluid cavities, specify the reference nodes associated with the primary and secondary fluid cavities. When the flow is based on analysis conditions, the fluid will flow from the high pressure or upstream cavity to the low pressure or downstream cavity and the heat energy will flow from the high temperature to the low temperature.

Input File Usage: Use the following options:

**FLUID CAVITY*, NAME=*primary_cavity_name*,
REF NODE=*primary_cavity_reference_node*
**FLUID CAVITY*, NAME=*secondary_cavity_name*,
REF NODE=*secondary_cavity_reference_node*
**FLUID EXCHANGE*, NAME=*fluid_exchange_name*
primary_cavity_reference_node, secondary_cavity_reference_node

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Definition**: **Between cavities**, **Fluid cavity interaction 1**: *name*, **Fluid cavity interaction 2**: *name*, **Fluid exchange property**: *name*

Specifying the Effective Area in an Abaqus/Explicit Analysis

The flow rate from the primary cavity for any fluid exchange property is proportional to the effective leakage area. The leakage area might represent the size of an exhaust orifice, the area of a porous fabric enclosing the cavity, or the size of a pipe between cavities.

In an Abaqus/Explicit analysis you can specify the value of the effective leakage area directly. Alternatively, you can define a surface that represents the leakage area by specifying the name of the surface on the boundary enclosing the primary fluid cavity. The effective area for fluid exchange is based on the area of the surface unless you specify the area directly or define the effective area with user subroutine *VUFLUIDEXCHEFFAREA*. If both the effective area and a surface are specified, the area of the surface is used only to determine blockage; see *Accounting for Blockage due to Contacting Boundary Surfaces* below. If neither area is specified, the effective area defaults to 1.0.

You can also define the effective leakage area with user subroutine *VUFLUIDEXCHEFFAREA* (see *VUFLUIDEXCHEFFAREA*) if leakage needs to be modeled as a function of the material state in the underlying

elements of the specified surface. For example, this subroutine can be used to define the leakage area at an element level for modeling fabric permeability in uncoated airbags where the leakage can vary locally depending on the strains in the yarn directions and the angle between the fabric yarns. Only membrane elements are supported for use with *VUFLUIDEXCHEFFAREA*.

Input File Usage: Use the following option to specify the effective leakage area directly and to specify a surface that represents the leakage area:

**FLUID EXCHANGE*, EFFECTIVE AREA=*effective_area*, SURFACE=*surface_name*

Use the following option to define the effective leakage area with a user subroutine:

**FLUID EXCHANGE*, EFFECTIVE AREA=USER, SURFACE=*surface_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Effective exchange area**: *effective_area*

User subroutine *VUFLUIDEXCHEFFAREA* is not supported in Abaqus/CAE.

Fluid Exchange through Ruptured Surfaces

Elements enclosing fluid cavities might fail and create a ruptured leakage area allowing for fluid exchange. For example, two fluid cavities that share a common wall modeled by membrane elements will not exchange fluid as long as the membrane elements remain intact. When any of the shared membrane elements fail, fluid will be exchanged through an effective area determined by the sum of the area of the failed elements. In essence, the failed elements become holes in the membrane through which fluid can flow.

In an Abaqus/Explicit analysis you can define a surface set whose underlying elements might fail and allow fluid and/or heat energy to be exchanged through the surfaces of the failed elements. The effective area for the fluid exchange is computed from the surfaces of the failed elements.

For a high pressure fluid chamber, such as a balloon, rupture of a small portion of the enclosing surface can completely destroy the fluid chamber. In this case you can choose to deactivate the fluid cavity by setting a maximum rupture area ratio. The area ratio is defined as the area of the surfaces from the failed elements over the total area of the user-defined surface set for the fluid exchange. Once the current rupture area ratio exceeds the specified maximum, the cavity pressure is no longer applied to the fluid cavity surfaces.

In a fluid cavity computation only the failure of the elements used to define the fluid cavity can be detected. If a fluid cavity is physically enclosed by multiple layers of elements, the failure of the immediately adjacent elements creates a leakage path for fluid exchange even though no physical path exists. In such cases, fluid exchange based on the surfaces of failed elements should be used with caution.

Input File Usage:

**FLUID EXCHANGE*, NAME=*fluid_exchange_name*, RUPTURE=ON, MAX RUPTURE AREA RATIO=*threshold ratio*

Abaqus/CAE Usage: Fluid exchange through ruptured surfaces is not supported in Abaqus/CAE.

Application of Fluid Cavity Pressure on a Fluid Exchange Surface

You can control how the effect of the cavity pressure on a fluid exchange surface is accounted for in Abaqus/Explicit. By default, the cavity pressure generates forces at all of the fluid exchange surface nodes, using the same method as for other portions of the fluid cavity. Optionally, the resultant force of the cavity pressure

on the fluid exchange surface can be distributed among only the nodes that lie on the perimeter of the fluid exchange surface (for example, of the nodes shown on the fluid exchange surface in *Figure 1*, only the nodes at locations A and B lie on the perimeter). This option can be used to avoid local bulging of a vent surface that will cause inaccurate computation of the leakage area. *Figure 2* shows an example of bulging when cavity pressure forces are distributed among all nodes of a vent surface.

In an Abaqus/Explicit analysis, when elements enclosing a fluid cavity fail, the fluid cavity pressure is not applied on the surfaces of those failed elements, which might help prevent potential numerical issues associated with free-flying nodes of failed elements enclosing the fluid cavity.

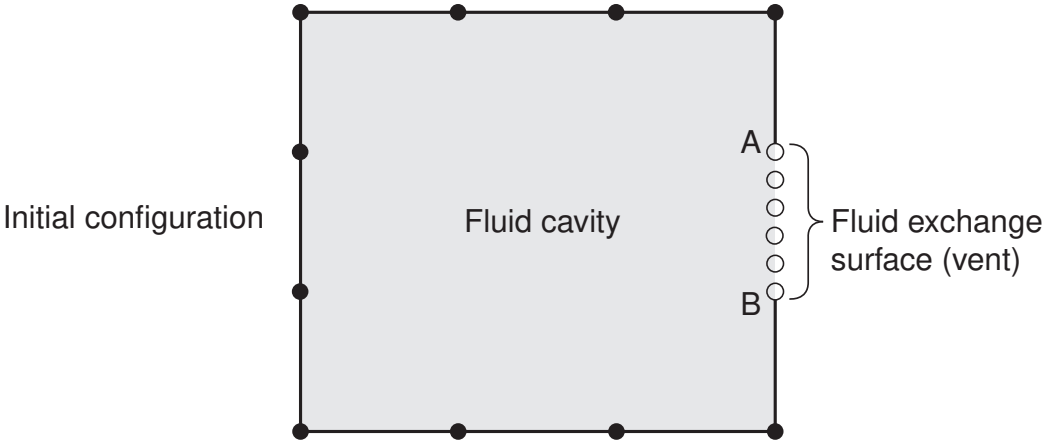


Figure 1: Initial configuration of a fluid exchange surface.

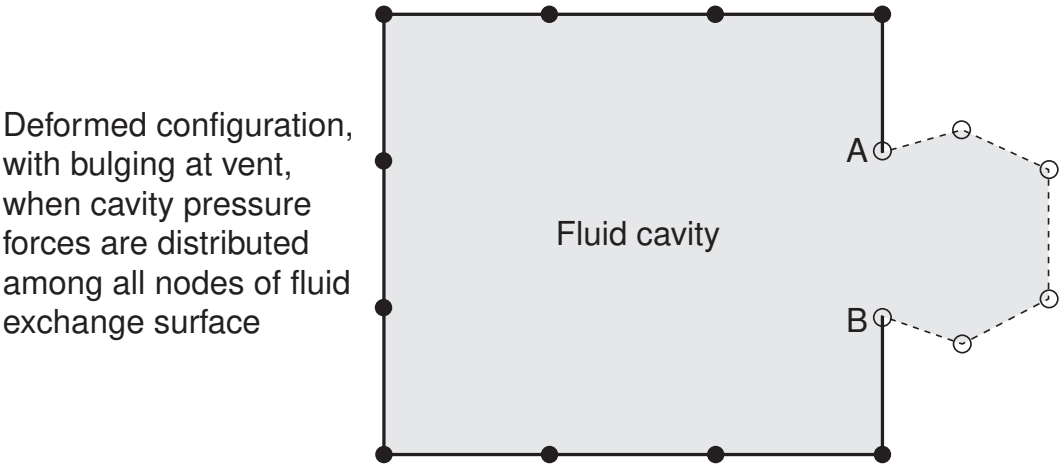


Figure 2: Deformed configuration of a fluid exchange surface.

Input File Usage: Use the following option (default) to indicate that the fluid pressure should generate forces on all nodes of a fluid exchange surface:

**FLUID EXCHANGE*, CAVITY PRESSURE=SURFACE, SURFACE=*surface_name*

Use the following option to indicate that the fluid pressure should generate force only on perimeter nodes of a fluid exchange:

**FLUID EXCHANGE*, CAVITY PRESSURE=PERIMETER, SURFACE=*surface_name*

Abaqus/CAE Usage: You cannot change the default pressure application in Abaqus/CAE. The pressure is always applied to all of the fluid exchange surface nodes.

Defining the Fluid Exchange Property

There are several different types of fluid exchange properties available in Abaqus to define the rate flow from a fluid cavity to the environment or between two cavities. The fluid exchange property can be as simple as prescribing the mass or volume flow rate directly. More complex leakage mechanisms such as those found on automotive airbags can be modeled by defining the mass or volume leakage rate as a function of the pressure difference, Δp; the absolute pressure, $\tilde{p}$; and the temperature, θ. The heat loss due to heat transfer through the surface of the cavity can be modeled in Abaqus/Explicit by prescribing the heat energy flow rate directly or by defining the heat energy flow rate as a function of the temperature difference, $\Delta\theta$; the absolute pressure, $\tilde{p}$; and the temperature, θ. Alternatively, in Abaqus/Explicit the mass flow rate and/or heat energy flow rate can be specified in user subroutine *VUFLUIDEXCH*.

For the purposes of evaluating the mass flow rate between two cavities, the absolute pressure and temperature are taken from the high pressure or upstream cavity. The mass flow is always in the direction from the high pressure cavity to the low pressure or downstream cavity, and the heat energy flow is always in the direction from the high temperature cavity to the low temperature cavity. The cavity absolute pressure and temperature are always used to calculate the flow between a cavity and the environment.

You must associate the fluid exchange property with a name. This name can then be used to associate a certain property with a fluid exchange definition.

Input File Usage: Use the following options:

**FLUID EXCHANGE*, NAME=*fluid_exchange_name*, PROPERTY=*property_name*
**FLUID EXCHANGE PROPERTY*, NAME=*property_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**, **Name**: *property_name*

Specifying a Mass or Volume Flux

Fluid flux into or out of the primary fluid cavity can be defined directly by prescribing the mass flow rate per unit area, $\dot{\bar{m}}$. The mass flow rate is

$$\dot{m} = \dot{\bar{m}} A,$$

where A is the effective area.

Fluid flux can also be defined by prescribing a volumetric flow rate per unit area, $\dot{\bar{V}}$. The mass flow rate is

$$\dot{m} = \rho \dot{\bar{V}} A,$$

where ρ is the density.

A negative value for $\dot{\bar{m}}$ or $\dot{\bar{V}}$ will generate flux into the primary fluid cavity. When a second fluid cavity is not defined, the state of the fluid flowing into the primary cavity is assumed to be that of the fluid already present in the primary cavity.

Input File Usage: To prescribe a flux based on mass flow rate:

**FLUID EXCHANGE PROPERTY*, TYPE=MASS FLUX

To prescribe a flux based on volumetric flow rate:

**FLUID EXCHANGE PROPERTY*, TYPE=VOLUME FLUX

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Mass flux** or **Volume flux**

Specifying the Flow Rate Using the Viscous and Hydrodynamic Resistance Coefficients

The mass flow rate, $\dot{m}$, can be related to pressure difference by both viscous and hydrodynamic resistance coefficients such as

$$\Delta pA = C_V \dot{m} + C_H \dot{m} |\dot{m}|,$$

where Δp is the pressure difference, A is the effective area, C_V is the viscous resistance coefficient, and C_H is the hydrodynamic resistance coefficient. The resistance coefficients can be functions of the average absolute pressure, average temperature, and average of any user-defined field variables. A positive value of $\dot{m}$ corresponds to flow out of the first cavity.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=BULK VISCOSITY, DEPENDENCIES=*n*
viscous resistance coefficient (C_V), hydrodynamic resistance coefficient (C_H)

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Bulk viscosity**: **Viscous coefficient**: C_V: **Hydrodynamic coefficient**: C_H

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Specifying the Flow Rate through a Vent or Exhaust Orifice

The mass flow rate through a vent or exhaust orifice that can be approximated by one-dimensional, quasi-steady, and isentropic flow is given (Bird, Stewart, and Lightfoot, 2002) by

$$|\dot{m}| = CA \frac{\tilde{p}_e}{\sqrt{R\left(\theta - \theta^Z\right)}} \sqrt{\frac{2\gamma}{\gamma - 1}\left(q^{\frac{2}{\gamma}} - q^{\frac{\gamma+1}{\gamma}}\right)},$$

where C is the dimensionless discharge coefficient, A is the vent or exhaust orifice area, θ is the temperature in the upstream fluid cavity, θ^Z is the absolute zero on the temperature scale being used, and $\tilde{p}_e$ is the absolute pressure in the upstream fluid cavity. The pressure ratio, q, is defined as

$$q = \frac{\tilde{p}}{\tilde{p}_e},$$

where $\tilde{p}$ is the absolute pressure in the orifice. The critical pressure, p_c, at which choked or sonic flow occurs is defined as

$$p_c = \tilde{p}_e \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma}{\gamma - 1}},$$

where γ is the ratio of the constant pressure heat capacity, c_p, and the constant volume heat capacity, c_v:

$$\gamma = \frac{c_p}{c_v}.$$

The orifice pressure, $\tilde{p}$, is then given by

$$\tilde{p} = p_a \quad \text{if } p_a \geq p_c,$$
$$\tilde{p} = p_c \quad \text{if } p_a < p_c,$$

where p_a is equal to the ambient pressure for flow out of a single fluid cavity or the downstream cavity pressure for flow between two fluid cavities.

The value of the discharge coefficient can be a function of the absolute upstream pressure, upstream temperature, and any user-defined field variables. Fluid exchange through a vent or exhaust orifice is valid only for pneumatic fluids and is available only in Abaqus/Explicit.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=ORIFICE, DEPENDENCIES=*n*
discharge coefficient

Abaqus/CAE Usage: Fluid exchange through vents or orifices is not supported in Abaqus/CAE.

Specifying the Flow Rate due to Fabric Leakage

The mass flow rate due to leakage through fabric can be expressed as

$$|\dot{m}| = CA\sqrt{2\rho\,|\Delta p|},$$

where C is the dimensionless fabric leakage or discharge coefficient and A is the effective fabric leakage area.

The value of the discharge coefficient can be a function of absolute upstream pressure, upstream temperature, and any user-defined field variables.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=FABRIC LEAKAGE, DEPENDENCIES=*n*
discharge coefficient

Abaqus/CAE Usage: Defining fluid exchange due to fabric leakage is not supported in Abaqus/CAE.

Specifying a Table of Mass Flow Rate Versus Pressure Difference

The overall mass flow rate can be calculated from a specified mass flow rate per unit area, $\dot{\bar{m}}$, by

$$|\dot{m}| = \dot{\bar{m}}\left(|\Delta p|, \tilde{p}, \theta\right) A,$$

where A is the effective area.

In this case you can define the mass flow rate per unit area in a table depending on the absolute value of pressure difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\bar{m}}$ and $|\Delta p|$ must be positive and start from zero.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=MASS RATE LEAKAGE, DEPENDENCIES=*n*
0, 0
$\dot{\bar{m}}_1$, $|\Delta p|_1$
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Mass rate leakage**: **Mass Flow Rate**: $\dot{\overline{m}}$, **Pressure Difference**: $|\Delta p|$

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Specifying a Table of Volumetric Flow Rate Versus Pressure Difference

The overall mass flow rate can be calculated from a specified volumetric flow rate per unit area, $\dot{\overline{V}}$, by

$$|\dot{m}| = \rho \dot{\overline{V}} \left(|\Delta p| , \tilde{p}, \theta \right) A,$$

where *A* is the effective area and ρ is the density.

In this case you can define the volumetric flow rate per unit area in a table depending on the absolute value of pressure difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\overline{V}}$ and $|\Delta p|$ must be positive and start from zero.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=VOLUME RATE LEAKAGE, DEPENDENCIES=*n*
0, 0
$\dot{\overline{V}}_1$, $|\Delta p|_1$
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Volume rate leakage**: **Volumetric Flow Rate**: $\dot{\overline{V}}$, **Pressure Difference**: $|\Delta p|_1$

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Specifying a Heat Energy Flux

In Abaqus/Explicit heat energy flux into or out of the primary fluid cavity can be defined directly by prescribing the heat energy flow rate per unit area, $\dot{\overline{Q}}$. The heat energy flow rate is

$$\dot{Q} = \dot{\overline{Q}} A,$$

where *A* is the effective area. A positive value for $\dot{\overline{Q}}$ generates heat flux out of the primary fluid cavity.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=ENERGY FLUX

Abaqus/CAE Usage: Defining fluid exchange by specifying the heat energy flow rate explicitly is not supported in Abaqus/CAE.

Specifying a Table of Heat Energy Flow Rate Versus Temperature Difference

The overall heat energy flow rate can be calculated from a specified heat energy flow rate per unit area, $\dot{\bar{Q}}$, by

$$\dot{Q} = \dot{\bar{Q}}\left(|\Delta\theta|, \tilde{p}, \theta\right) A,$$

where A is the effective area.

In this case in Abaqus/Explicit you can define the heat energy flow rate per unit area in a table depending on the absolute value of temperature difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\bar{Q}}$ and $|\Delta\theta|$ must be positive and start from zero.

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=ENERGY RATE LEAKAGE, DEPENDENCIES=*n*
0, 0
$\dot{\bar{Q}}_1$, $|\Delta\theta|_1$
...

Abaqus/CAE Usage: Defining fluid exchange by specifying the heat energy flow rate as a function of temperature difference and pressure is not supported in Abaqus/CAE.

Specifying Mass Flow Rate and/or Heat Energy Flow Rate with a User Subroutine

The mass flow rate, $\dot{m}$, or the overall heat energy flow rate, $\dot{Q}$, can be defined in Abaqus/Explicit using user subroutine *VUFLUIDEXCH* (see *VUFLUIDEXCHEFFAREA*).

Input File Usage: **FLUID EXCHANGE PROPERTY*, TYPE=USER

Abaqus/CAE Usage: User subroutine *VUFLUIDEXCH* is not supported in Abaqus/CAE.

Activating the Fluid Exchange Definition in Abaqus/Explicit

Fluid exchange will not occur in Abaqus/Explicit unless the fluid exchange definition is activated in an analysis step.

Input File Usage: Use the following options to activate a fluid exchange for a given analysis step:

**FLUID EXCHANGE*, NAME=*fluid_exchange_name*
**FLUID EXCHANGE ACTIVATION*
fluid_exchange_name

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange activation**

Varying the Magnitude of the Flow

By default, the magnitude of the flow is based on the specified flow behavior. A time variation of flow magnitude during a step can be introduced by an amplitude curve. The magnitude based on the specified flow behavior is

multiplied by the amplitude value to obtain the actual mass or heat energy flow rate. For example, a time variation of prescribed mass or volumetric flux can be defined.

An amplitude curve might be used to trigger an event for fluid exchange in the middle of a step. For example, an airbag might deploy at some predetermined time during a step, and it might be desirable to close off all exhaust orifices until the actual deployment. A step amplitude curve that starts at zero and steps up at deployment time could be used for this purpose.

Input File Usage: Use the following options:

**AMPLITUDE*, NAME=*amplitude_name*
**FLUID EXCHANGE ACTIVATION*, AMPLITUDE=*amplitude_name*

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid exchange activation**, **Amplitude**: *amplitude_name*

Accounting for Blockage due to Contacting Boundary Surfaces

Abaqus/Explicit can account for the blockage of flow out of a cavity due to an obstruction caused by contacting surfaces. For example, flow out of an exhaust orifice might be fully or partially blocked because it is covered by another contacting surface.

Blockage can be considered for any fluid exchange property. However, a surface must be defined on the boundary of the fluid cavity to be checked for contact obstruction. Abaqus/Explicit will calculate the area fraction of the surface not blocked by contacting surfaces and apply this fraction to the mass or energy flow rate out of the cavity. You can control the combination of surfaces that can cause blockage. Abaqus/Explicit will not consider contacting surfaces to cause blockage unless you specify that they can potentially cause blockage (see *Contact Blockage*).

Input File Usage: **FLUID EXCHANGE ACTIVATION*, BLOCKAGE=YES

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid exchange activation**, **Consider blockages by contacted surface**

Limiting the Flow Direction

By default, flow can occur both in and out of the primary fluid cavity when a second node is included in the fluid exchange definition. In addition, heat energy flow can occur in both directions when flow is defined between a single cavity and its environment. You can limit the flow direction in Abaqus/Explicit in these cases such that fluid or heat energy flows only out of the primary fluid cavity. This method is relevant only for a fluid exchange definition based on analysis conditions and not on prescribed mass, volume, or heat energy flux.

Input File Usage: **FLUID EXCHANGE ACTIVATION*, OUTFLOW ONLY

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid exchange activation**, **Allow outflow only**

Activating the Fluid Exchange Based on the Change in the Leakage Area

The flow between cavities can be activated in Abaqus/Explicit based on a change in the area of the surface defining the effective area. You need to specify the ratio of the actual surface area to the initial effective area, which represents the threshold value for triggering the fluid exchange. The effective area used for the fluid

exchange between the cavities (or between the cavity and the ambient) is the area difference between the actual area and the initial area.

Input File Usage: Use the following options:

**FLUID EXCHANGE*, SURFACE=*surface_name*
**FLUID EXCHANGE ACTIVATION*, DELTA LEAKAGE AREA=*surface_ratio*

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid exchange activation**, **Delta leakage area**

Activation in Multiple Steps

By default, when you modify the activation of a fluid exchange definition or activate a new fluid exchange definition, all existing fluid exchange activations in the step remain. When modifying an existing activation, all applicable data must be respecified.

Activated fluid exchange definitions remain active in subsequent steps unless deactivated. You can choose to deactivate all fluid exchange definitions in the model and optionally reactivate new ones. If you deactivate any fluid exchange definition in a step, all fluid exchange definitions must be respecified.

Input File Usage: Use the following option to modify an existing fluid exchange activation or to specify an additional fluid exchange activation (default):

**FLUID EXCHANGE ACTIVATION*, OP=MOD

Use the following option to deactivate all fluid exchange definitions in the model and optionally reactivate new ones:

**FLUID EXCHANGE ACTIVATION*, OP=NEW

Abaqus/CAE Usage: Use the following option to modify an existing fluid exchange activation:

Interaction module: **Edit Interaction** or **Interaction Manager**: **Edit** specific fluid exchange activation interaction

Use the following option to deactivate fluid exchange definitions:

Interaction module: **Interaction Manager**: **Deactivate** specific fluid exchange activation interaction

Specifying Mass Flux in Abaqus/Standard

In Abaqus/Standard the amount of fluid in a cavity can be varied in a step. An amplitude curve can be used to define the mass flow rate during the particular step.

Input File Usage: Use the following options:

**AMPLITUDE*, NAME=*amplitude_name*
**FLUID FLUX*, AMPLITUDE=*amplitude_name*

Use the following option to modify an existing fluid flux or to specify an additional fluid flux to a cavity (default):

**FLUID FLUX*, OP=MOD

Use the following option to deactivate all fluid flux definitions in the model and optionally reactivate new ones:

**FLUID FLUX*, OP=NEW

Abaqus/CAE Usage: The use of fluid flux to modify mass flow rates is not supported in Abaqus/CAE.

References

- Bird, R. B., W. E. Stewart, and E. N. Lightfoot, Transport Phenomena, Wiley, New York , 2002.

Inflator Definition

Products: Abaqus/Explicit Abaqus/CAE

References:

- *About Surface-Based Fluid Cavities*
- *Fluid Cavity Definition*
- *Fluid Exchange Definition*
- **FLUID INFLATOR*
- **FLUID INFLATOR PROPERTY*
- **FLUID INFLATOR ACTIVATION*
- *Defining a fluid inflator interaction*
- *Defining a fluid inflator interaction property*

Overview

An inflator definition:

- can be used to inflate a fluid cavity to simulate actual inflators used for airbag supplemental restraint systems;
- can inflate a fluid cavity with an ideal gas mixture different from that present in the fluid cavity;
- can be specified directly or by defining data from a tank test;
- has a name that can be used to identify history output of mass flow rates; and
- can be activated at any time during the analysis.

Defining an Inflator

The inflator capability in Abaqus/Explicit is suited for modeling the flow characteristics of inflators used for airbag systems. You must associate the inflator definition with a name. You specify the reference node of the fluid cavity that the inflator will fill with gas. A single fluid cavity can have any number of inflators.

Input File Usage: **FLUID INFLATOR*, NAME=*name*
fluid_cavity_reference_node

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid inflator**, **Name**: *name*; **Fluid cavity interaction**: *name*

Input File Usage: **FLUID INFLATOR*, TYPE=COMBINED
fluid_cavity_reference_node, Eulerian_material_instance
inflator_nodes

Defining the Inflator Property

The inflator property defines the mass flow rate and temperature as a function of inflation time either directly or by entering tank test data. It also defines the mixture of gases entering the fluid cavity. You must associate

the inflator property with a name. This name can then be used to associate a certain property with an inflator definition.

Input File Usage: Use the following options:

**FLUID INFLATOR*, NAME=*fluid_inflator_name*, PROPERTY=*property_name*
**FLUID INFLATOR PROPERTY*, NAME=*property_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid inflator**, **Name**: *property_name*

Specifying the Gas Temperature and Mass Flow Rate Directly

The temperature and the mass flow rate of the gas entering the fluid cavity can be given directly as functions of inflation time. Enter a table of mass flow rate and temperature versus inflation time.

Input File Usage: **FLUID INFLATOR PROPERTY*, TYPE=TEMPERATURE AND MASS
inflation time, inflator gas temperature, inflator mass flow rate
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid inflator**, **Definition**: **Temperature and mass**: **Inflation Time**, **Inflator Gas Temperature**, **Inflator Mass Flow Rate**

Using Tank Test Data

The mass flow rate and the temperature of the gas entering the fluid cavity can be determined by the results of a tank test. In the test the inflator is discharged into a closed, fixed volume tank, and the time history of pressure in the tank is measured. The inflator mass flow rate can then be calculated from the pressure history using the equations of gas dynamics. For an ideal gas, conservation of energy for an adiabatic process is given by

$$\dot{m}_{tank} c_v \theta_{tank} + m_{tank} c_v \dot{\theta}_{tank} = \dot{m}_{in} c_p \left(\theta_{in} - \theta^z\right),$$

where θ is the temperature, θ^z is the absolute zero on the temperature scale being used, and the subscripts in and $tank$ refer to quantities in the inflator and the rigid tank, respectively. Using mass balance

$$\dot{m}_{tank} = \dot{m}_{in},$$

and the equation of state for an ideal gas with constant volume gives

$$\dot{p}_{tank} V_{tank} = R\left(\dot{m}_{tank} \theta_{tank} + m_{tank} \dot{\theta}_{tank}\right).$$

The mass flow rate can be found by combining the above equations

$$\dot{m}_{in} = \frac{\dot{p}_{tank} V_{tank}}{\gamma R \left(\theta_{in} - \theta^z\right)},$$

where γ is the ratio of the constant pressure heat capacity, c_p, and the constant volume heat capacity, c_v:

$$\gamma = \frac{c_p}{c_v}.$$

To calculate the mass flow rate using the results of a tank test, enter a table of tank pressure and inflator temperature versus inflation time, and specify the volume of the tank.

Input File Usage: **FLUID INFLATOR PROPERTY*, TYPE=TANK TEST, TANK VOLUME=V_{tank}
inflation time, inflator gas temperature, tank pressure
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid inflator**, **Definition**: **Tank test**: **Tank volume**: V_{tank}, **Inflation Time**, **Inflator Gas Temperature**, **Tank Pressure**

Using the Dual Pressure Method

If both the inflator pressure, $\tilde{p}_{in}$, and tank pressure, p_{tank}, time history curves can be measured during a tank test, the inflator mass flow rate and temperature can then be calculated using the assumption of isentropic flow (Wang and Nefske, 1988). The mass flow rate through the inflator orifice can be described by

$$\dot{m}_{in} = CA\frac{\tilde{p}_{in},}{\sqrt{R\left(\theta_{in} - \theta^{Z}\right)}}C_{tank},$$

where C is the discharge coefficient, A is the effective area, and the coefficient C_{tank} is determined by assuming choked or sonic flow as

$$C_{tank} = \left(\frac{2}{\gamma+1}\right)^{\frac{1}{\gamma-1}}\sqrt{\frac{2\gamma}{\gamma+1}}.$$

Comparing the expression for inflator mass flow rate obtained in a rigid tank with that given above, the inflator temperature is given by

$$\theta_{in} - \theta^{Z} = \frac{1}{R}\left(\frac{\dot{p}_{tank}V_{tank}}{\gamma CAC_{tank}\tilde{p}_{in}}\right)^{2},$$

and the inflator mass flow rate is

$$\dot{m}_{in} = \frac{\gamma\left(CAC_{tank}\tilde{p}_{in}\right)^{2}}{V_{tank}\dot{p}_{tank}}.$$

To calculate the inflator mass flow rate and temperature using the dual pressure method, enter a table of tank pressure and inflator pressure versus inflation time; and specify the volume of the tank, the effective area, and the discharge coefficient. The tank volume and effective area must be specified. The discharge coefficient has a default value of 0.4.

Input File Usage: **FLUID INFLATOR PROPERTY*, TYPE=DUAL PRESSURE,
TANK VOLUME=V_{tank}, EFFECTIVE AREA=A,
DISCHARGE COEFFICIENT=C
inflation time, inflator pressure, tank pressure
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid inflator**, **Definition**: **Dual pressure**: **Effective area**: A, **Tank volume**: V_{tank}, **Discharge coefficient**: C, **Inflation Time**, **Inflator Pressure**, **Tank Pressure**

Specifying the Inflator Pressure and Mass Flow Rate Directly

You can enter a table of the mass flow rate and inflator pressure versus inflation time and specify the effective area and discharge coefficient. The gas temperature in the inflator will be calculated by using the assumption of isentropic flow. The effective area must be specified. The discharge coefficient has a default value of 0.4.

Input File Usage: **FLUID INFLATOR PROPERTY*, TYPE=PRESSURE AND MASS, EFFECTIVE AREA=*A*, DISCHARGE COEFFICIENT=*C*
inflation time, inflator pressure, inflator mass flow rate
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid inflator**, **Definition**: **Inflator pressure and mass flow rate**: **Effective area**: *A*, **Discharge coefficient**: *C*, **Inflation Time**, **Inflator Pressure**, **Inflator Mass Flow Rate**

Specifying the Gas Mixture

To define the inflator gas mixture, specify the number of gas species used for the inflator, and enter a list of names of fluid behaviors and a table of the mass fraction or molar fraction of the species. The mass fraction or molar fraction of the species might be a function of inflation time. The sum of the mass fractions or molar fractions for the species should be equal to one at any given time.

Input File Usage: Use the following options to specify the gas mixture in terms of the mass fractions:

**FLUID INFLATOR PROPERTY*
**FLUID INFLATOR MIXTURE*, NUMBER SPECIES=*k*,
TYPE=MASS FRACTION
fluid_behavior_name_1, fluid_behavior_name_2, etc.
inflation time, mass fraction 1, mass fraction 2, etc.
...

Use the following options to specify the gas mixture in terms of the molar fractions:

**FLUID INFLATOR PROPERTY*
**FLUID INFLATOR MIXTURE*, NUMBER SPECIES=*k*,
TYPE=MOLAR FRACTION
fluid_behavior_name_1, fluid_behavior_name_2, etc.
inflation time, molar fraction 1, molar fraction 2, etc.
...

Abaqus/CAE Usage: Use the following options to specify the gas mixture in terms of the mass fractions:

Interaction module: **Create Interaction Property**: **Fluid Inflator Mixture**, **Type**: **Mass fraction**

Use the following options to specify the gas mixture in terms of the molar fractions:

Interaction module: **Create Interaction Property**: **Fluid Inflator Mixture**, **Type**: **Molar fraction**

Activating the Inflator Definition

Inflation does occur unless the inflation definition is activated in an analysis step.

Input File Usage: Use the following options to activate a fluid inflator for a given analysis step:

**FLUID INFLATOR*, NAME=*fluid_inflator_name*
**FLUID INFLATOR ACTIVATION*
fluid_inflator_name

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid inflator activation**

Relating Inflation Time to Analysis Time

Inflator property definition consists of specifying tables of gas variables versus inflation time. In Abaqus/Explicit the inflation time, t_{in}, is related to the value of an amplitude curve $f(t)$ by

$$t_{in} = \int_0^t f(t)\, dt.$$

Typically the amplitude variation is a step function stepping from zero to one at the time the airbag should be deployed. This amplitude variation has the effect of offsetting the inflation time from the analysis time.

Input File Usage: Use the following options:

**AMPLITUDE*, NAME=*amplitude_name*
**FLUID INFLATOR ACTIVATION*, INFLATION TIME
AMPLITUDE=*amplitude_name*

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid inflator activation**, **Inflation time amplitude**: *amplitude_name*

Modifying the Mass Flow Rate

If the mass flow rate is prescribed directly in the inflator property definition, you can modify it by specifying an amplitude definition during a step. However, if the mass flow rate is calculated using tank test data or the dual pressure method, the amplitude definition will be ignored.

Input File Usage: Use the following options:

**AMPLITUDE*, NAME=*amplitude_name*
**FLUID INFLATOR ACTIVATION*, MASS FLOW AMPLITUDE=*amplitude_name*

Abaqus/CAE Usage: Interaction module: **Edit Interaction**: **Fluid inflator activation**, **Mass flow amplitude**: *amplitude_name*

Activation in Multiple Steps

By default, when you modify the activation of a fluid inflator definition or activate a new fluid inflator definition, all existing fluid inflator activations in the step remain. When modifying an existing activation, all applicable parameters must be respecified.

Activated inflator definitions remain active in subsequent steps unless deactivated. You can choose to deactivate all fluid inflator definitions in the model and optionally reactivate new ones. If you deactivate any fluid inflator definition in a step, all fluid inflator definitions must be respecified.

Input File Usage: Use the following option to modify an existing fluid inflator activation or to specify an additional fluid inflator activation (default):

**FLUID INFLATOR ACTIVATION*, OP=MOD

Use the following option to deactivate all fluid inflator definitions in the model and optionally reactivate new ones:

**FLUID INFLATOR ACTIVATION*, OP=NEW

Abaqus/CAE Usage: Use the following option to modify an existing fluid inflator activation:

Interaction module: **Edit Interaction** or **Interaction Manager**: **Edit** specific fluid inflator activation interaction

Use the following option to deactivate all fluid inflator definitions in the model:

Interaction module: **Interaction Manager**: **Deactivate** specific fluid inflator activation interaction

References

- Wang, J. T., and O. J. Nefske, "A New CAL3D Airbag Inflation Model," *SAE paper 880654*, 1988.

Mass Scaling

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Explicit Dynamic Analysis*
- *Mass Adjustment*
- *About Output*
- **FIXED MASS SCALING*
- **VARIABLE MASS SCALING*
- *Configuring a dynamic, explicit procedure*
- *Configuring a dynamic fully coupled thermal-stress procedure using explicit integration*

Overview

You can use element mass scaling for computational efficiency and contact mass scaling to avoid reductions in the time increment size due to contact stiffness.

Element mass scaling is often used in Abaqus/Explicit for computational efficiency in quasi-static analyses and in some dynamic analyses that contain a few very small elements that control the stable time increment. You can use element mass scaling to:

- scale the mass of the entire model or scale the masses of individual elements and/or element sets;
- scale the mass on a per step basis in a multistep analysis; and
- scale the mass at the beginning of the step and/or throughout the step.

You can use contact mass scaling to avoid reductions in the time increment size due to contact stiffness. Contact mass scaling can be used together with or separately from element mass scaling. Contact mass scaling adds mass to surface nodes to avoid having the local time increment drop below the global time increment that would be in effect without consideration of the contact stiffness.

Introduction

The explicit dynamics procedure is typically used to solve two classes of problems: transient dynamic response calculations and quasi-static simulations involving complex nonlinear effects (most commonly problems involving complex contact conditions). Because the explicit central difference method is used to integrate the equations in time (see *Explicit Dynamic Analysis*), the discrete mass matrix used in the equilibrium equations plays a crucial role in both computational efficiency and accuracy for both classes of problems. When used appropriately, mass scaling can often improve the computational efficiency while retaining the necessary degree of accuracy required for a particular problem class. However, the mass scaling techniques most appropriate for quasi-static simulations might be very different from those that should be used for dynamic analyses.

Quasi-Static Analysis

For quasi-static simulations incorporating rate-independent material behavior, the natural time scale is generally not important. To achieve an economical solution, it is often useful to reduce the time period of the analysis or to increase the mass of the model artificially (“mass scaling”). Both alternatives yield similar results for rate-independent materials, although mass scaling is the preferred means of reducing the solution time if rate dependencies are included in the model because the natural time scale is preserved.

Mass scaling for quasi-static analysis is usually performed on the entire model. However, when different parts of a model have different stiffness and mass properties, it may be useful to scale only selected parts of the model or to scale each of the parts independently. In any case, it is never necessary to reduce the mass of the model from its physical value, and it is generally not possible to increase the mass arbitrarily without degrading accuracy. A limited amount of mass scaling is usually possible for most quasi-static cases and results in a corresponding increase in the time increment used by Abaqus/Explicit and a corresponding reduction in computational time. However, you must ensure that changes in the mass and consequent increases in the inertial forces do not alter the solution significantly.

Although mass scaling can be achieved by modifying the densities of the materials in the model, the methods described in this section offer much more flexibility, especially in multistep analyses.

See *Rolling of thick plates* for a discussion of using mass scaling in a quasi-static analysis.

Dynamic Analysis

The natural time scale is always important in dynamic analysis, and an accurate representation of the physical mass and inertia in the model is required to capture the transient response. However, many complex dynamic models contain a few very small elements, which will force Abaqus/Explicit to use a small time increment to integrate the entire model in time. These small elements are often the result of a difficult mesh generation task. By scaling the masses of these controlling elements at the beginning of the step, the stable time increment can be increased significantly, yet the effect on the overall dynamic behavior of the model may be negligible.

During an impact analysis, elements near the impact zone typically experience large amounts of deformation. The reduced characteristic lengths of these elements result in a smaller global time increment. Scaling the mass of these elements as required throughout the simulation can significantly decrease the computation time. For cases in which the compressed elements are impacting a stationary rigid body, increases in mass for these small elements during the simulation will have very little effect on the overall dynamic response.

Mass scaling for truly dynamic events should almost always occur only for a limited number of elements and should never significantly increase the overall mass properties of the model, which would degrade the accuracy of the dynamic solution.

See *Impact of a copper rod* for a discussion of using mass scaling in a dynamic analysis.

Stable Time Increments

Throughout this section the term “element stable time increment” refers to the stable time increment of a single element. The term “element-by-element stable time increment” refers to the minimum element stable time increment within a specific element set. The term “stable time increment” refers to the stable time increment of the entire model, regardless of whether the global estimator or the element-by-element estimator is used.

Mass Scaling for Contact

You can use contact mass scaling to avoid reductions in the time increment size due to contact stiffness. In the absence of contact mass scaling, the default contact stiffness in Abaqus/Explicit might cause the time increment to decrease by up to 5%. The time increment reduction can become much more significant if a larger contact stiffness is in effect. For example, the time increment reduction might be up to 34% if the contact stiffness is 10 times greater than the default contact stiffness. Contact mass scaling adds mass to surface nodes to avoid having the local time increment drop below the global time increment that would be in effect without consideration of contact stiffness.

Introducing Element Mass Scaling into a Model

Two types of element mass scaling are available in Abaqus/Explicit: fixed mass scaling and variable mass scaling. You can apply these two types of element mass scaling along with contact mass scaling (*Mass Scaling to Account for Contact Stiffness*) separately, or you can apply them together to define an overall mass scaling strategy.

Element mass scaling can also apply globally to the entire model or, alternatively, on an element set–by– element set basis.

Element mass scaling can be performed by:

- scaling the masses of all specified elements by a user-supplied constant factor;
- scaling the masses of all specified elements by the same value so that the minimum stable time increment for any element in the element set is equal to a user-supplied time increment;
- scaling the masses of only the elements in the element set whose element stable time increments are less than a user-supplied time increment so that the element stable time increment for these elements becomes equal to the user-supplied time increment;
- scaling the masses of all specified elements so that their element stable time increments each become equal to the user-supplied time increment; and
- scaling automatically based on mesh geometry and initial conditions for bulk metal rolling analyses.

Fixed Mass Scaling

Fixed mass scaling is performed once at the beginning of the step for which it is specified. Two basic approaches are available for fixed mass scaling: you can define a mass scaling factor directly, or you can define a desired minimum stable time increment for which the mass scaling factors are determined by Abaqus/Explicit.

If both variable mass scaling and fixed mass scaling are specified in a step, the element original mass is scaled once at beginning of that step based on the specified fixed mass scaling. It is then further scaled at the beginning and periodically during that step based on the specified variable mass scaling.

Fixed mass scaling provides a simple means to modify the mass properties of a quasi-static model at the beginning of an analysis or to modify the masses of a few small elements in a dynamic model so that they do not control the stable time increment size. Since the scaling operation is performed only once at the beginning of the step for which the mass scaling is defined, fixed mass scaling is computationally efficient.

Input File Usage: **FIXED MASS SCALING*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**

Variable Mass Scaling

Variable mass scaling is used to scale the mass of elements at the beginning of a step and periodically during that step. When using this type of mass scaling, you define a desired minimum stable time increment: mass scaling factors will be calculated automatically and applied, as required, throughout the step.

If both variable mass scaling and fixed mass scaling are specified in a step, the element original mass is scaled once at beginning of that step based on the specified fixed mass scaling. It is then further scaled at the beginning and periodically during that step based on the specified variable mass scaling.

Variable mass scaling is most useful when the stiffness properties that control the stable time increment change drastically during a step. This situation can occur in both quasi-static bulk forming and dynamic simulations in which elements are highly compressed or crushed.

Input File Usage: **VARIABLE MASS SCALING*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**

Defining a Scale Factor Directly

Defining a scale factor directly is useful for quasi-static analyses in which the kinetic energy in the model should remain small. You can define a fixed mass scaling factor that is applied to the original mass of all elements in a specified element set. The masses of the elements will be scaled at the beginning of the step and held fixed throughout the step unless further modified by variable mass scaling.

Input File Usage: **FIXED MASS SCALING*, FACTOR=*scale_factor*

For example, the following option scales the masses of elements contained in element set *elset* by a factor of 10:

**FIXED MASS SCALING*, FACTOR=10., ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**, **Scale by factor:** *scale_factor*

Defining a Desired Element-by-Element Stable Time Increment

You can define a desired element-by-element stable time increment for an element set for fixed or variable mass scaling. Abaqus/Explicit will then determine the necessary mass scaling factors. There are three mutually exclusive methods available to scale the mass of the model when a desired element-by-element stable time increment is defined. Each method is described in detail later in this section.

To determine the stable time increment used during an increment, Abaqus/Explicit first determines the smallest stable time increment on an element-by-element basis. Then, a global estimation algorithm determines a stable time increment based on the highest frequency of the model. The larger of the two estimates determines the stable time increment used. In general, the stable time increment determined by the global estimator will be greater than the stable time increment determined by the element-by-element estimator. When fixed or variable mass scaling is used with a specified element-by-element stable time increment to scale the mass of a set of elements, the element-by-element stable time increment estimate is being affected directly. If all of the elements in the model are being scaled by a single mass scaling definition, the element-by-element estimate will equal the value assigned to the element-by-element stable time increment unless the penalty method is being used to enforce contact constraints. Penalty contact can cause the element-by-element estimate to be slightly below the value assigned to the element-by-element stable time increment (see *Contact Controls for General Contact in Abaqus/Explicit* and *Contact Constraint Enforcement Methods in Abaqus/Explicit*). The actual stable time increment used may be greater than the value assigned to the element-by-element stable time increment because of the use of the global estimator. If mass scaling is performed on only a portion of the model, the elements that are not scaled may have element stable time increments that are less than the value assigned to the element-by-element stable time increment and in that case will control the element-by-element stable time increment estimate. As a result, if only portions of the model are being scaled, the time increment used will generally not equal the value assigned to the element-by-element stable time increment.

If the fixed time increment size for the explicit dynamic step is based on the initial element-by-element stability limit or is specified directly, the time increment used will be calculated according to the rules described in *Explicit Dynamic Analysis*.

Scaling the Mass Uniformly

Scaling the mass uniformly is useful for quasi-static analyses in which the kinetic energy in the model should remain small. This approach is similar to defining a scale factor directly. In both cases the masses of all the elements specified are scaled uniformly by a single factor. However, with this method the mass scaling factor is determined by Abaqus/Explicit instead of being user specified. A single mass scaling factor is applied uniformly to all the elements so that the minimum stable time increment within these elements is equal to the value assigned to the element-by-element stable time increment, *dt*.

Input File Usage: Use either of the following options:

**FIXED MASS SCALING*, TYPE=UNIFORM, DT=*dt*
**VARIABLE MASS SCALING*, TYPE=UNIFORM, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: Uniformly to satisfy target**

Scaling Only Elements with Element Stable Time Increments below the Specified Element-by-Element Stable Time Increment

Scaling elements with element stable time increments below a user-specified value is appropriate for both quasi-static and dynamic analyses. It is useful for increasing the element stable time increment of the most critical elements.

When the mesh at the beginning of an analysis or a step contains a few very small elements that control the stable time increment size, use fixed mass scaling to scale the masses of those elements and start the step with a desired time increment value. Increasing the mass of only these controlling elements means that the stable time increment can be increased significantly, yet the effect on the overall behavior of the model may be negligible.

For analyses in which evolving deformation creates a limited number of small elements, use variable mass scaling to scale the masses of those elements, thereby limiting the reduction in the stable time increment.

Input File Usage: Use either of the following options:

**FIXED MASS SCALING*, TYPE=BELOW MIN, DT=*dt*
**VARIABLE MASS SCALING*, TYPE=BELOW MIN, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: If below minimum target**

Scaling All Elements to Have Equal Element Stable Time Increments

Scaling all elements such that they have the same stable time increment effectively contracts the eigenspectrum of the model; that is, it reduces the range between the lowest and highest natural frequency of the model. Because of the drastic change in mass properties, this approach is appropriate only for quasi-static analyses. It implies that some elements may have mass scaling factors that are less than one.

Input File Usage: Use either of the following options:

*FIXED MASS SCALING, TYPE=SET EQUAL DT, DT=*dt*
*VARIABLE MASS SCALING, TYPE=SET EQUAL DT, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: Nonuniformly to equal target**

Mass Scaling to Account for Contact Stiffness

Abaqus/Explicit uses the penalty method to enforce contact constraints, as discussed in *Penalty Contact Algorithm*. Contact stiffness (and damping) acting in addition to element stiffness (and damping) on surface nodes tends to increase the highest natural frequency of the system, and, therefore, decrease the stable time increment size (*Stability*). Abaqus/Explicit provides options to scale the mass of surface nodes to account for contact.

Default contact penalty stiffness and damping in Abaqus/Explicit are such that they contribute to at most a 5% reduction in the stable time increment. However, specifying a higher penalty stiffness can cause more significant reductions in the stable time increment in the absence of contact mass scaling. For example, increasing the contact stiffness by a factor of 10 might cause up to a 34% reduction in the stable time increment.

Contact mass scaling adds mass to surface nodes to avoid having the local time increment drop below the global time increment that would be in effect without consideration of contact stiffness. Element mass scaling might affect the global time increment that would be in effect without consideration of contact stiffness. The amount of additional mass required to avoid reductions in the time increment due to contact is often quite small, because only surface nodes actively involved in contact are involved in contact mass scaling. You can limit contact mass scaling to surface nodes potentially involved in element mass scaling or you can apply it to all surfaces nodes (including for nodes not associated with element mass scaling and for models without any element mass scaling).

Input File Usage: Use the following option to apply contact mass scaling only to surface nodes potentially involved in element mass scaling (default):

*CONTACT MASS SCALING, LOCATION=ELEMENT MASS SCALING

Use the following option to apply contact mass scaling to all the surface nodes involved in the general contact:

*CONTACT MASS SCALING, LOCATION=ALL CONTACT SURFACES

Use the following option to apply contact mass scaling only on the contact surfaces defined in the data lines:

*CONTACT MASS SCALING, LOCATION=SPECIFIED SURFACES
surf_1, *surf_2*, ...

Use the following option to avoid contact mass scaling:

*CONTACT MASS SCALING, LOCATION=NONE

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Contact Mass Scaling, Explicit**

Global and Local Mass Scaling

Specifying an element set for either fixed or variable mass scaling scales the mass of a localized region of the model. Omitting an element set implies that mass scaling will be performed for all elements. A global definition can be overwritten by a local definition for a given element set by repeating the mass scaling definition with an element set specified.

Input File Usage: Use either of the following options:

**FIXED MASS SCALING*, ELSET=*elset*
**VARIABLE MASS SCALING*, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Region: Set:** *elset*

Example 1

Different mass scaling factors may be useful when materials with vastly different wave speeds or mesh refinements are present in an analysis. In this example a scale factor of 50 may be desirable for the masses of all elements in a quasi-static analysis, except for a few elements for which a mass scaling factor of 500 is used.

```
*FIXED MASS SCALING, FACTOR=50.0
*FIXED MASS SCALING, FACTOR=500.0, ELSET=elset1
```

The first fixed mass scaling definition scales the masses of all elements in the model by a factor of 50. The second fixed mass scaling definition overrides the first definition for the elements contained in element set *elset1* by scaling their masses by a factor of 500.

Example 2

An alternative method of scaling the masses of elements in *elset1* is to assign a stable time increment to them and allow Abaqus/Explicit to determine the mass scaling factors.

```
*FIXED MASS SCALING, FACTOR=50.0
*FIXED MASS SCALING, DT=.5E-6, TYPE=BELOW MIN, ELSET=elset1
```

The first fixed mass scaling definition scales the masses in the entire model by a factor of 50. The second fixed mass scaling definition overrides the first definition by scaling the masses of any elements in *elset1* whose stable time increments are less than $.5 \times 10^{-6}$.

Mass Scaling at the Beginning of the Step

Fixed mass scaling is used to prescribe mass scaling only at the beginning of a step and always scales the original element masses. When the scale factor is defined directly, the mass is scaled by the value assigned to the scale factor. If the element-by-element stable time increment, dt, is specified, the mass scaling is based on this value. If both the scale factor and the element-by-element stable time increment are specified, the mass is first scaled by the value assigned to the scale factor and then possibly scaled again, depending on the value assigned to the element-by-element stable time increment and the type of fixed mass scaling chosen.

Local mass scaling can be defined for a specific element set. If no element set is specified, the fixed mass scaling definition will apply to all elements in the model. Only one fixed mass scaling definition is permitted per element set. Multiple fixed mass scaling definitions cannot contain overlapping element sets. Local mass scaling definitions will overwrite global definitions for the specified element sets.

Input File Usage: **FIXED MASS SCALING*, FACTOR=*factor*, DT=*dt*, TYPE=*type*, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**, **Scale by factor:** *factor*, **Scale to target time increment of:** *dt*

Example

Assume that for a quasi-static analysis a mass scaling factor of 50 is applied to all the elements in the model. Furthermore, assume that even after being scaled by a factor of 50, a few extremely small or poorly shaped elements are causing the stable time increment to be less than a desired minimum. To increase the stable time increment, the following option is used:

```
*FIXED MASS SCALING, FACTOR=50., TYPE=BELOW MIN, DT=.5E-6
```

The specified scale factor causes the masses of all the elements in the model to be scaled by a factor of 50. If any element's stable time increment is still below 0.5×10^{-6} after being scaled by a factor of 50.0, its mass will be scaled such that its stable time increment is equal to 0.5×10^{-6}.

Mass Scaling throughout the Step

Variable mass scaling with a specified element-by-element stable time increment is used to define mass scaling that is to be performed at the beginning and throughout the step. Either the frequency in increments or the number of intervals must be specified to define how frequently mass scaling is to be performed. In increments other than those in which mass scaling is performed, the time increment used will generally be different from the value assigned to the element-by-element stable time increment.

Local mass scaling can be defined for a specific element set. If no element set is specified, the variable mass scaling definition will apply to all elements in the model. Only one variable mass scaling definition is permitted per element set. Multiple variable mass scaling definitions cannot contain overlapping element sets. Local mass scaling definitions will overwrite global definitions for the specified element sets.

Input File Usage: **VARIABLE MASS SCALING*, DT=*dt*, TYPE=*type*, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*

Calculating the Mass Scaling at Equally Spaced Increments

You can specify the number of increments between mass scaling calculations. For example, specifying a frequency of 5 will cause mass scaling to be performed at the beginning of the step and at increments 5, 10, 15, etc.

Care should be taken when choosing the value of the frequency, since performing mass scaling every few increments during an analysis may result in noticeable additional computational cost per increment.

Input File Usage: **VARIABLE MASS SCALING*, TYPE=*type*, DT=*dt*, FREQUENCY=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*, **Scale: Every *n* increments**

Calculating the Mass Scaling at Equally Spaced Time Intervals

Alternatively, you can specify the number of equally spaced time intervals at which the mass scaling calculations are to be performed. For example, specifying 5 intervals in a step with a duration of one second will cause mass scaling to be performed at the beginning of the step and at times of .2 , .4, .6, .8, and 1.0 seconds.

Input File Usage: **VARIABLE MASS SCALING*, TYPE=*type*, DT=*dt*, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*, **Scale: At *n* equal intervals**

Different Mass Scaling at the Beginning and during the Step

There are cases where it is desirable to include mass scaling at the beginning of a step that may be modified further throughout the step.

Input File Usage: Use both of the following options:

**FIXED MASS SCALING*, FACTOR=*factor*, TYPE=*type*, DT=*dt_init*
**VARIABLE MASS SCALING*, TYPE=*type*, DT=*dt_min*, FREQUENCY=*n*
or NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Create both of the following mass scaling definitions:

Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**:
Semi-automatic mass scaling, **Scale: At beginning of step**
Semi-automatic mass scaling, **Scale: Throughout step**

Example

Assume that in a dynamic impact analysis, a few extremely small or poorly shaped elements exist in the mesh and consequently control the stable time increment. To prevent these elements from controlling the stable time increment, it is desirable to scale their masses at the beginning of the step. In addition, elements in a region of the mesh will develop severe distortions as a result of impact with a fixed rigid surface. Consequently, elements in the impact zone may eventually control the stable time increment.

Since the elements in the impact zone are essentially stationary against the rigid surface, selectively scaling their masses will guarantee that the overall dynamic response is not adversely affected. Mass scaling these elements by prescribing a time increment to limit the reduction in the element-by-element stable time increment may decrease run time substantially.

For example, specify fixed mass scaling for all elements in the model with stable time increments below a value of 1.0×10^{-6}. In addition, specify variable mass scaling for the elements in the impact zone (*elset1*) with stable time increments below a value of 0.5×10^{-6}. In this case all the elements in the model are checked at the beginning of the step. If any have stable time increments less than 1.0×10^{-6}, their masses are scaled (independently) such that the element-by-element stable time increment equals 1.0×10^{-6}. This scaling remains in effect throughout the step and is not further modified, except for those elements in *elset1*. The variable mass scaling definition causes the elements contained in *elset1* to be scaled throughout the step so that their stable time increments do

not become less than 0.5×10^{-6}. Because only elements in *elset1* are scaled during the step, it is possible that a stable time increment less than 0.5×10^{-6} may result.

Mass Scaling in a Multiple Step Analysis or an Abaqus/Explicit to Abaqus/Explicit Import Analysis

When mass scaling is applied, Abaqus/Explicit automatically carries forward the following to the subsequent step or import analysis:

- the scaled element masses at the end of one step or a previous analysis
- any variable mass scaling methods specified in that step or analysis

This approach ensures that continuity is carried forward automatically to the subsequent step or the import analysis. This ensures continuity in the mass matrix at the subsequent step/analysis and continued application of the variable mass scaling methods. However, you can reset the element masses to their original values or recompute the element masses by using a new fixed mass scaling method at the beginning of the subsequent step or the import analysis. You can also remove the variable mass scaling methods inherited from the prior step/analysis or replace an inherited method with a new variable mass scaling method.

To reset the initial mass matrix, specify a fixed mass scaling method in the subsequent step or in the import analysis. Similarly, specify a variable mass scaling method in the subsequent step or in the import analysis to discontinue all of the variable mass scaling methods of the prior step/analysis. The examples below illustrate the following special cases: continuous mass matrix with no further mass scaling, and reverting the mass matrix to the original state with no further mass scaling.

Very large changes in element mass across the steps due to mass scaling may lead to precision problems in the mass calculations. These precision problems may give rise to erroneous or misleading results. When large changes in element masses are desired in such situations, it is recommended that fixed mass scaling be used in the new step or in the import analysis to reset the element masses to their original values before using additional mass scaling definitions, as required, to scale the element masses to their desired values.

Continuous Mass Matrix with No Further Scaling

To define a continuous mass matrix with no further scaling, remove any variable mass scaling definitions inherited from the prior step/analysis by redefining a new variable mass scaling definition.

Input File Usage: Use the following option without any parameters in a new step or an import analysis:

**VARIABLE MASS SCALING*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Disable mass scaling throughout step**

Example

Assume that during the first step of a quasi-static analysis, elements experience distortions that cause the stable time increment to decrease dramatically. Furthermore, assume that the deformation during the second step is not large enough to have any further effect on the stable time increment. Similarly, assume the same behavior for an analysis followed by an import analysis.

```
*HEADING
…
*STEP
…
*FIXED MASS SCALING, FACTOR=1.1
*VARIABLE MASS SCALING, TYPE=BELOW MIN, DT=1.E-5, FREQUENCY=10
```

```
…
*END STEP
*STEP
…
*VARIABLE MASS SCALING
…
*END STEP
```

During the first step/analysis the fixed mass scaling increases the element mass by the factor 1.1. The variable mass scaling definition scales the mass of the entire model at the beginning of the step and every tenth increment such that the element-by-element stable time increment equals at least 1×10^{-5}. The variable mass scaling definition in the second step or the import analysis replaces the one continued from the first step/analysis. This particular definition of variable mass scaling without any parameters in the second step/analysis also prevents any further mass scaling during the second step/analysis. The scaled mass matrix from the first step/analysis is carried over to be used during the entire second step/analysis.

Reverting the Mass Matrix to the Original State

You can introduce a fixed mass scaling method in the subsequent step or the import analysis to discontinue all of the mass scaling methods of the prior step/analysis. Further, if the default specification of fixed mass scaling is used, element masses revert to their original values at the beginning of the subsequent step or the import analysis. Therefore, specify just the default fixed mass scaling method to prevent the scaled mass of the previous step/analysis from being used in a new step or an import analysis. This approach is useful when you are going from a quasi-static simulation step/analysis where mass scaling is appropriate to a dynamic step/analysis in which no scaling is desired.

Input File Usage: Use both of the following options without any parameters:

**FIXED MASS SCALING*
**VARIABLE MASS SCALING*

Abaqus/CAE Usage: Create both of the following mass scaling definitions:

Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**:
Reinitialize mass
Disable mass scaling throughout step

Example

Assume that an analysis contains a quasi-static step followed by a dynamic step, or assume a quasi-static analysis is followed by a dynamic analysis. Mass scaling can be performed during the quasi-static step/analysis but turned off during the dynamic step/analysis.

```
*HEADING
*STEP
…
*FIXED MASS SCALING, FACTOR=1.1
*VARIABLE MASS SCALING, TYPE=BELOW MIN, DT=1.E-5, FREQUENCY=10
*END STEP
*STEP
*FIXED MASS SCALING
*VARIABLE MASS SCALING
*END STEP
```

During the first step/analysis the fixed mass scaling increases the element mass by the factor 1.1. The variable mass scaling definition scales the mass of the entire model at the beginning of the step and every tenth increment such that the element-by-element stable time increment equals at least 1×10^{-5}. The new fixed mass scaling definition without any parameters in the second step or in the import analysis then reverts the mass matrix back to the original state. The new variable mass scaling definition replaces all the variable mass scaling definitions inherited from the first step/analysis. Further, since the new variable mass scaling definition has no parameters, no mass scaling is applied during the second step/analysis. Therefore, the mass matrix for the second step or the import analysis reverts to that of the original state.

Mass Contribution from External Programs Connected to Abaqus via Co-Simulation

Co-simulation can lead to mass and/or rotary inertia from external programs being added to the Abaqus model during a step. However, that contribution along with other quantities imported from the external program must be removed once the co-simulation step is completed. If co-simulation is expected to add mass and/or rotary inertia to the Abaqus model, Abaqus automatically reverts the mass matrix back to the original state once such a co-simulation step is completed. You need to respecify any mass scaling that must be continued beyond the co-simulation step.

When Mass Scaling Is or Is Not Used

This section discusses when mass scaling is or is not used for element mass scaling and contact mass scaling.

Element Mass Scaling

The following entities are not affected by element mass scaling:

- Thermal solution response in a fully coupled thermal-stress analysis
- Gravity loads, viscous pressure loads
- Adiabatic heat calculations
- Equation of state materials
- Fluid and fluid link elements
- Surface-based fluid cavities
- Spring and dashpot elements

Densities associated with any of the relevant items in this list will remain unscaled. Mass, rotary inertia, infinite, and rigid elements can be scaled. However, because none of the elements has an associated stable time increment, they can be scaled only using either a user-specified scale factor or an element-by-element stable time increment applied uniformly. If the element-by-element stable time increment is specified, at least one element with a stable time increment must be included in the element mass scaling definition. Rotary inertia in shell, beam, and pipe elements is based on the scaled mass.

The mass of infinite elements can be scaled; however, the infinite elements will not act as quiet boundaries unless the densities of each adjacent deformable element are scaled by the same factor. The mass of both elements will be scaled by the same factor if they are both included in the same fixed or variable element mass scaling definition.

Contact Mass Scaling

The following entities are not affected by contact mass scaling:

- Coupled Eulerian-Lagrangian analysis (CEL)
- Particle methods (SPH, DEM, and LKM)

Automatic Mass Scaling for Analysis of Bulk Metal Rolling

Bulk metal rolling is generally considered a quasi-static process, but the process is often modeled with Abaqus/Explicit because of its ability to handle the contact problem well. To achieve an economical solution with Abaqus/Explicit, it is often useful to increase the mass of the product artificially. However, the mass scaling factor must be chosen such that the changes in the mass and the corresponding changes in the inertial forces do not alter the solutions significantly. Choosing too high a scaling factor will not produce quasi-static results. Choosing too low a scaling factor, while conservative, will result in long run times. Rolling variable mass scaling can be used to make the choice of the optimal scaling factor automatic for this process.

The automatic strategy is based on the semi-automatic method of scaling all elements to have equal element stable time increments. The method is made automatic by determining the appropriate value for the target stable time increment from several parameters of the rolling process. The value used for the target stable time increment, Δt, is based on the average element length in the rolling direction, L_e; the feed rate, V; and the number of nodes in the cross-section of the product, n. The feed rate is defined as the average velocity of the product in the rolling direction during steady-state conditions. The value of Δt is adjusted during the analysis to account for the actual value of the feed rate. You must specify estimated values for the average velocity, the average element length in the rolling direction, and the number of nodes in the cross-section of the product.

The mass of any element will never drop below its original mass. This is different from the method of scaling all elements to have equal element stable time increments. Imposing this restriction means that rolling problems that have significant inertial effects will not have their mass adjusted automatically when they are analyzed as quasi-static.

To achieve a good result, it is recommended that:

- the product be meshed by extruding a two-dimensional cross-section of the product;
- the average element length in the rolling direction not vary significantly along the length of the product;
- the product have an initial velocity in the rolling direction approximately equal to the steady-state feed rate;
- the element size in the cross-section be equal to or less than the size in the rolling direction; and
- no other mass scaling be used on elements scaled with rolling automatic variable mass scaling.

Input File Usage: **VARIABLE MASS SCALING*, ELSET=*elset1*, FREQUENCY=*n*,
TYPE=ROLLING, FEED RATE=*V*, EXTRUDED LENGTH=L_e,
CROSS SECTION NODES=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Automatic mass scaling**, **Feed rate:** *V*, **Extruded element length:** L_e, **Nodes in cross section:** *n*

Output

You can request the following output variables to support the contribution from element mass scaling:

- Output variable EMSF provides the element mass scaling factor. You can use Abaqus/CAE to obtain contour and history plots of EMSF.
- Output variable EDT provides the element stable time increment. The element stable time increment includes the effect of mass scaling. You can use Abaqus/CAE to obtain history plots of EDT.

You can request the following output variables to support the contribution from contact mass scaling:

- Output variable CNMSF provides the nodal contact mass scaling factor.
- Output variable CDMASS provides the total percent change in mass of the model as a result of contact mass scaling.

You can request the following output variables to support the contribution from element and contact mass scaling:

- Output variable DMASS provides the total percent change in mass of the model as a result of contact and element mass scaling and is available for history plotting in Abaqus/CAE. Output variable DMASS is not available on an element set basis.
- Output variable NDTTOTAL provides the stable time increment over contact surfaces with the effects of element and contact mass scaling.

Selective Subcycling

Products: Abaqus/Explicit

References:

- *Explicit Dynamic Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- **SUBCYCLING*

Overview

Selective subcycling:

- allows different time increments to be used for different groups of elements;
- reduces run time for an analysis when a small region of elements in the model controls the stable time increment; and
- is invoked by defining the subcycling zones.

Introduction

The selective subcycling method in Abaqus/Explicit is based on domain decomposition. In this method subcycling zones are defined that remain unchanged during the analysis. The domain-level parallelization method (*Parallel Execution in Abaqus/Explicit*) is invoked automatically when subcycling zones are defined. Each subcycling zone, as well as the nonsubcycling zone, is independently decomposed into the user-specified number of parallel domains. The "nonsubcycling" domains are defined as the parallel domains that are derived from the nonsubcycling zone and are integrated with the largest stable time increment. The remaining parallel domains derived from the subcycling zones are integrated using smaller time increments, or "subcycles."

The subcycle time increment sizes are chosen as integer divisors of the time increment used in the nonsubcycling parallel domains. Therefore, all parallel domains exactly reach the same time points as the nonsubcycling parallel domains. During subcycling, nodes that lie on the interface with the nonsubcycling zone require special treatment. The velocity at the interface nodes is taken from the nonsubcycling zone and is constant during subcycles. This produces an interface node displacement field that varies linearly during the subcycles.

Defining Subcycling Zones

Subcycling zones are defined by element sets. You can include all element types in these sets except EC3D8R, EC3D8RT, C3D10, C3D10T, PD3D, and PC3D elements. However, all parallel domains must have at least one deformable element to provide the stable time increment. Abaqus/Explicit issues an error message if there is no deformable element in a parallel domain. You can define an arbitrary number of subcycling zones. However, some modeling features cannot be split between subcycling zones. Abaqus/Explicit automatically merges subcycling zones that contain features that cannot be split. Subcycling zones are merged together when:

- the zones overlap;
- the zones share the same nodes;
- a node is in one subcycling zone, but its adjacent nodes are in a different subcycling zone;
- subcycling zones are involved in the same constraint equation, connector, or rigid body; or

- general contact is specified in the analysis.

When subcycling zones are merged, the smallest stable time increment among the merged zones is used. The constraint, connector, or rigid body is always assigned to the subcycling zone if any one of its nodes is involved in that subcycling zone. Since the domain-level parallelization method is used, all restrictions on parallel domain decomposition apply to subcycling zones. These restrictions prevent certain features from being split across nonsubcycling parallel domains, as well as parallel domains that contain the subcycling zones (See *Parallel Execution in Abaqus/Explicit*). Beam elements with exact rotary inertia cannot share nodes between subcycling and nonsubcycling zones. Analytical rigid surfaces cannot be included in the general contact domain when a subcycling zone is defined.

Efficient selective subcycling requires proper choice of subcycling zones. For each subcycling zone, the time increment size should be small compared to the nonsubcycling zone, producing a large number of subcycles. The number of subcycles is the ratio of the stable time increment size in the nonsubcycling zone to the stable time increment sizes in the subcycling zones. In addition to a large number of subcycles, the number of elements in a subcycling zone should generally be small compared to the total number of elements in the model for optimal performance benefit. If a majority of elements in the model are in subcycling zones, there will not be much performance benefit.

Input File Usage: Use the following option to define a subcycling zone:

**SUBCYCLING*, ELSET=*element_set_name*

Accuracy of Results

The subcycling algorithm used in Abaqus/Explicit provides sufficient accuracy for most complex dynamic models. However, because of the relatively large time increment size used in the nonsubcycling zone and the interpolation used on zone interface nodes, subcycling solutions can introduce a truncation error, which may slightly alter results compared with traditional solutions. This error should not affect the overall dynamic behavior of the model. Special attention should be given to the interface between the subcycling zone and nonsubcycling zone when general contact (see *About General Contact in Abaqus/Explicit*) is involved. It is not necessary to define a pair of surfaces that have the potential for contacting each other within the same zone. However, to minimize truncation errors, it is highly recommended that a single surface that has the potential for contacting others not be split across the zones.

Output and Mass Scaling

Output (see *About Output*) and mass scaling (see *Mass Scaling*) are always performed at the same time points reached by all parallel domains.

Input File Template

```
*HEADING
 …
*ELSET, ELSET=ZONE1
 …
*SUBCYCLING, ELSET=ZONE1
**********************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
...
*END STEP
```

Steady-State Detection

Products: Abaqus/Explicit

References:

- *About Output*
- **STEADY STATE DETECTION*
- **STEADY STATE CRITERIA*

Overview

Steady-state detection:

- can be used to detect the time in a quasi-static unidirectional Abaqus/Explicit simulation when a steady-state condition has been reached and then terminate the simulation;
- can be used to output quantities that are useful in tracking the progress of a unidirectional Abaqus/Explicit simulation; and
- is available only for three-dimensional analysis.

Introduction

Many types of unidirectional processes are used to transform preformed shapes into forms more suitable for further processing. The most common examples are rolling, wire drawing, and extrusion processes. Since the processes are usually carried out at low speeds, explicit dynamic procedures such as those in Abaqus/Explicit are often used to model the processes as quasi-static. The analyses usually consist of a workpiece that is formed into a desired shape by any number of rollers or other forming surfaces along a primary direction. The forming surfaces are usually modeled as rigid bodies. For rolling simulations the rigid body reference node is usually defined at the center of the roller. The mesh of the workpiece is often extruded and may be constructed of multiple layers of material. As the workpiece progresses through the forming surfaces, the shape eventually reaches a constant state. The position where the workpiece exits the final forming surface is referred to as the exit plane and is usually aligned with the rigid body reference node of the final forming surface. As soon as this constant shape is reached, the analysis is considered to have reached steady state. The force and torque on the final forming surfaces at this steady-state condition have also reached constant values or oscillate about constant values. A significant computational savings can be achieved by detecting the steady-state condition and halting the analysis either immediately or as soon as the steady-state cross-section progresses beyond the exit plane to a position referred to as the cutting plane.

Mesh Requirements

The workpiece mesh is required to meet certain conditions for use with the steady-state detection capability. First, the mesh must be topologically regular in the primary direction. In other words, the mesh should consist of multiple planes of elements with each plane being similar to its adjacent leading and trailing planes in that it contains the same number of elements and the same element topology in the cross-section. Furthermore, each element in a plane is connected to elements in leading and trailing planes that reference the same material and section properties. Therefore, meshes with multiple materials and section properties are permitted, but any row of elements in the primary direction must be of the same type and must reference the same material and section properties (see *Figure 1*).

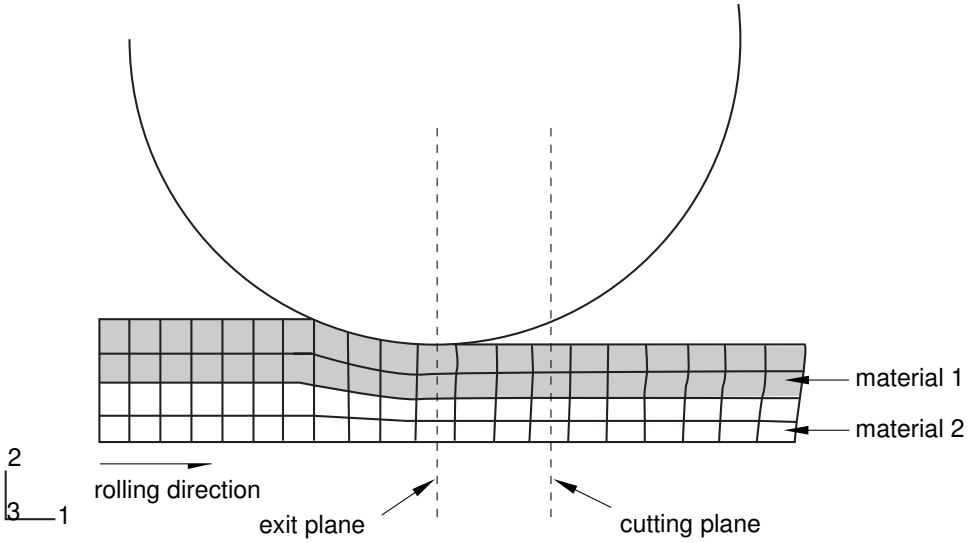


Figure 1: Acceptable multiple-material extruded mesh for a rolling analysis.

Steady-State Detection Criteria Sampling

To determine if steady state has been reached, steady-state detection “norms” are calculated, which represent an averaged value of a variable of interest over the cross-section of the workpiece as material passes through a given position along the primary direction. This position is referred to as the exit plane and usually coincides with the position of the last rigid forming tool (e.g., roller) that the workpiece passes through. The normal of the exit plane is by definition coincident with the primary direction. The time intervals at which the norms are sampled vary depending on whether the rolling analysis is modeled in an Eulerian or Lagrangian manner.

Sampling in a Lagrangian Analysis

In a Lagrangian-based analysis (which may include adaptive meshing employed on a Lagrangian domain) the steady-state norms are calculated as the trailing control node of each plane of elements passes the exit plane. *Figure 2* illustrates the control node definitions.

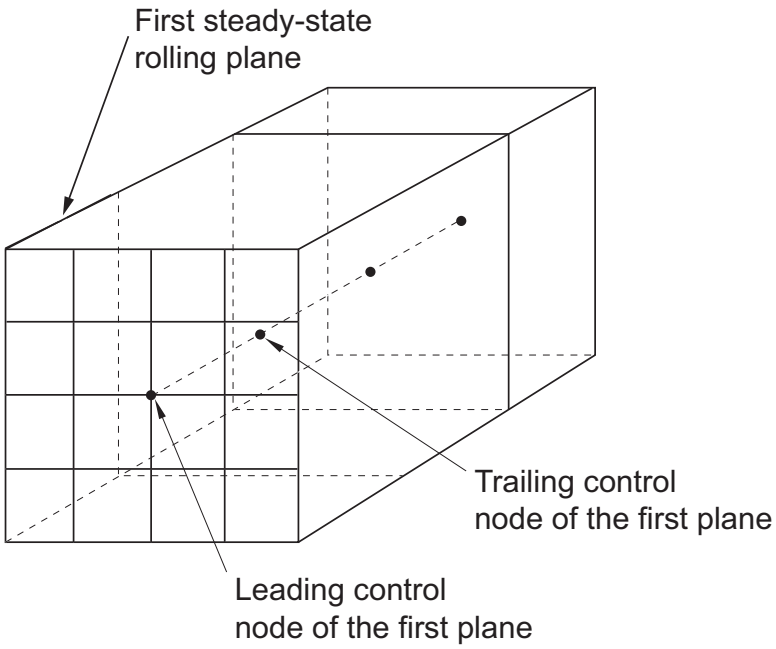


Figure 2: Control node positioning.

The time period of norm sampling is, therefore, based on the frequency at which the planes of elements cross the exit plane. For output purposes the values of the norms are assumed to remain constant between the times at which successive control nodes pass the exit plane.

Sampling in an Eulerian Analysis

An Eulerian analysis employs a control volume approach in which material is drawn from an inflow Eulerian boundary and is pushed or pulled out through an outflow boundary. Adaptive mesh domains are defined on the workpiece, and sliding boundary regions are defined to model contact between the workpiece and forming tools such as rollers. See *About ALE Adaptive Meshing* for details of adaptive meshing techniques. The mesh remains relatively stationary while the material moves through the exit plane. The time period between sampling is, therefore, based on the progress of the material moving through the exit plane. To determine a time period in a manner consistent with the Lagrangian case, the sampling period is determined by dividing the characteristic element length of the workpiece by the speed of the material flow. This period is roughly the time it takes for material to pass through an element of typical size.

Steady-State Detection Norm Definitions

An individual norm is considered to have achieved steady state if its relative change in value over three consecutive planes does not exceed a tolerance. You can provide the norm tolerances when you define the steady-state criteria, or default values of tolerances can be chosen by Abaqus/Explicit. The norms can be output by requesting their identifiers listed in the definitions below.

Equivalent Plastic Strain Norm

The plastic strain norm of a plane of elements is defined by summing the product of the equivalent plastic strain and the element volume of each element on the plane, then dividing by the total volume of the elements on the plane. This norm provides a weighted average of the equivalent plastic strain for the plane. The identifier for the equivalent plastic strain norm is SSPEEQ.

Spread Norm

The spread norm of a plane of elements is computed as the largest of the area moments of inertia of the cross-section of the plane. In determining the spread norm, the cross-section of the plane of elements is determined by projecting the element faces whose normals originally coincided with the primary direction onto the exit plane. The area moments of inertia are then determined about the centroid of the section in the directions of the original principal axes of the cross-section. The identifier for the spread norm is SSSPRD.

Force Norm

The force norm is computed by averaging the magnitude of the force at the rigid body reference node of a forming tool, such as the exit roller, over the time period between sampling points. You provide the rigid body reference node and force direction. The identifier for the force norm is SSFORC.

Torque Norm

The torque norm is computed by averaging the magnitude of the torque at the rigid body reference node of a forming tool, such as the exit roller, over the time period between sampling points. You provide the rigid body reference node and torque direction. The identifier for the torque norm is SSTORQ.

Requesting Steady-State Detection during an Analysis

You must define the criteria that are used to determine if steady state has been reached. Abaqus/Explicit will halt the analysis based on the achievement of steady state.

Steady-State Detection

A steady-state detection definition is used to define the elements in the workpiece, the primary direction of the workpiece, the cutting position, and the type of sampling used. The primary direction is defined by specifying the direction cosines with respect to the global Cartesian coordinate system. The cutting position is defined by specifying the global coordinates of a point lying in the cutting plane. The normal to the cutting plane is assumed to coincide with the primary direction. Once steady state has been detected, the analysis is terminated when the plane of the workpiece that was first detected to have reached steady state has progressed to the cutting plane. You can choose the sampling method used, as described below.

Requesting Sampling as Elements Pass the Exit Plane for a Lagrangian Analysis

You can request that all steady-state norms be calculated as each plane of elements crosses the exit plane.

Input File Usage: **STEADY STATE DETECTION*, ELSET=*elset*,
SAMPLING=PLANE BY PLANE

Requesting Sampling at Uniform Intervals for an Eulerian Analysis

Alternatively, you can request that all steady-state norms be calculated at an interval based on the time required for material to progress the length of an average element.

Input File Usage: **STEADY STATE DETECTION*, ELSET=*elset*, SAMPLING=UNIFORM

Steady-State Criteria

Any number of steady-state criteria definitions can be specified. Only when all of the criteria specified under any one steady-state criteria definition have been satisfied will the analysis be considered to have reached steady state.

To define the criteria, you specify the norm type identifier, the norm tolerance, and the global coordinates of a point on the exit plane. For force and torque norms, you also specify the rigid body reference node of the forming tool at the exit plane and the direction cosines of the force or torque. Exit planes can be defined separately for each norm definition.

Input File Usage: Use the following options to define the criteria needed to achieve steady state:

**STEADY STATE DETECTION*, ELSET=*elset*,
SAMPLING=PLANE BY PLANE or UNIFORM
**STEADY STATE CRITERIA*
**STEADY STATE CRITERIA*
...

For example, assume that two sets of criteria are of interest and that the analysis can be terminated as soon as either is satisfied. The input might be as follows:

**STEADY STATE DETECTION*, ELSET=sheet,
SAMPLING=PLANE BY PLANE
1.0, 0.0, 0.0, 6.0, 0.0, 0.0
**STEADY STATE CRITERIA*
SSPEEQ,.002, 5.0, 0.0, 0.0
SSSPRD,.002, 5.0, 0.0, 0.0
SSFORC,.005, 5.0, 0.0, 0.0, 1000, 1.0, 0.0, 0.0
SSFORC,.005, 5.0, 0.0, 0.0, 1000, 0.0, 1.0, 0.0
SSTORQ,.005, 5.0, 0.0, 0.0, 1000, 0.0, 0.0, 1.0
**STEADY STATE CRITERIA*
SSPEEQ,.001, 5.0, 0.0, 0.0
SSSPRD,.001, 5.0, 0.0, 0.0
SSFORC,.010, 5.0, 0.0, 0.0, 1000, 0.0, 1.0, 0.0

Materials

Steady-state detection is intended to be used with plasticity-based materials since the equivalent plastic strain norm would be zero for nonplasticity-based material models.

Procedures

One steady-state detection definition is allowed per analysis. The definition can be entered in any step and is continued through subsequent steps in an analysis. A steady-state detection definition cannot be entered in an annealing step or continued through an annealing step.

Elements

The current steady-state detection capabilities support the use of C3D8R and C3D8RT elements only.

Output

The output variables SSPEEQ*n*, SSSPRD*n*, SSFORC*n*, and SSTORQ*n* are used to output the equivalent plastic strain, spread, force, and torque norms, respectively. Abaqus/CAE can be used to obtain history plots of each of the steady-state detection norm variables. Individual norms can be output by requesting the norm number *n*, which is based on the order in which the norms are specified. Referring to the example above, if the force norm of the second steady-state criteria definition were to be requested, the output identifier would be SSFORC3. If a steady-state detection norm is requested that does not include a norm number, SSFORC for example, all norms of that type are output.

Once steady state has been detected, an element set is created automatically by Abaqus/Explicit and written to the output database consisting of the plane of elements that first satisfied the steady-state criteria. The element set created is named *SteadyStatePlane-StepN*, where *N* is the step number; and it can be viewed with Abaqus/CAE. If no output requests are made to the output database, the element set *SteadyStatePlane-StepN* is not created.

Input File Template

```
*HEADING
 …
*ELSET, ELSET=WORKPIECE
************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
...
*STEADY STATE DETECTION, ELSET=WORKPIECE, SAMPLING=PLANE BY PLANE
Data line specifying rolling direction and cutting plane position
*STEADY STATE CRITERIA
Data lines specifying steady-state detection norm criteria
...
*OUTPUT, HISTORY, TIME INTERVAL=1.E-6
*INCREMENTATION OUTPUT
SSPEEQ, SSSPRD, SSFORC, SSTORQ
...
*END STEP
```

CZone Analysis

Products: Abaqus/Explicit Abaqus/CAE

References:

- **CRUSH STRESS*
- **CRUSH STRESS VELOCITY FACTOR*
- **SURFACE PROPERTY ASSIGNMENT*
- *CZone Crush Stress*
- *Progressive Damage and Failure*
- *Shell Elements*

Overview

CZone for Abaqus is an Abaqus/Explicit capability that integrates material, element, and contact modeling aspects to simulate crushing of laminated composites due to contact with other bodies.

The CZone for Abaqus capability:

- requires specification of a “crush stress” within the material definition for parts that may undergo crushing;
- requires the use of shell elements to model thin parts that may undergo crushing;
- uses general contact to model contact interactions;
- limits the contact stress based on consideration of the underlying material;
- avoids nonphysical contact with deeply punctuated regions in an active crushing zone that can occur due to limiting the contact stress; and
- triggers element deletion due to crushing.

In part, this capability uses methodologies developed by Engenuity Limited (see *https://www.engenuity.net/*).

Introduction

The use of composites in automotive, aircraft, marine, military, and other equipment requires composites to be designed for predictable behavior both under normal loads and under extreme conditions. For example, energy absorption and failure of composite automotive body panels as they are crushed help protect the vehicle’s occupants in an accident. A rectangular cross-section cone designed for energy absorption upon impact is shown in *Figure 1*. Most of the composite cone crushes during the experiment to stop the test sled. *Figure 1* also shows a small portion of the cone remaining intact at the end of the experiment.

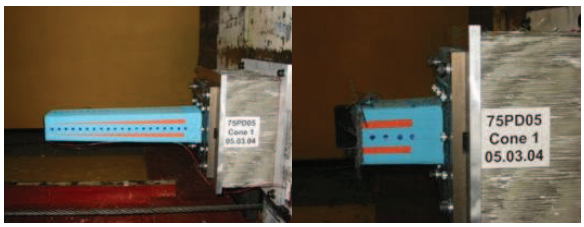


Figure 1: The plain cone test setup (left) and the plain cone sample at test completion (right) (courtesy of Engenuity, Ltd.).

The crushing behavior of composites is a complex finite element analysis problem that generally cannot be described by conventional failure mechanisms designed to simulate metal components. The material layers, reinforcements, and bonding between layers in a composite each have different failure characteristics. The load-carrying capacity of the composite is controlled by localized buckling, shearing, delamination, and eventual failure of the composite material in the area being crushed. While a material is crushed, it continues to transmit stress in a zone of damage called the crush front or crush zone, immediately adjacent to the point of impact between the composite and the object that is crushing it. Not all materials can be crushed. Physical testing is required to determine whether a material can be crushed and the characteristics of its behavior while being crushed.

The CZone for Abaqus capability integrates material, element, and contact algorithms to simulate continuous crushing behavior. *Figure 2* compares sled acceleration histories for the experiment associated with *Figure 1*. The experimental and simulation acceleration curves match well, with the simulation results smoother than the experimental data. Conventional material damage and element failure mechanisms for finite element analysis tend to produce noisy acceleration and stress solutions for crushing phenomena, due to sequences of element failure followed by a short interval of no contact until additional elements of the crushing body are impacted. Conventional approaches typically underestimate energy absorption associated with crushing, unlike the CZone approach.

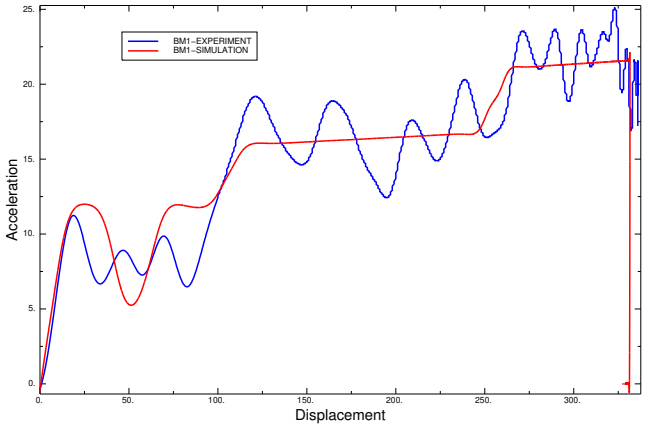


Figure 2: History of the sled acceleration for CZone for Abaqus simulation and experiment.

CZone Methodology

Figure 3 shows a simple example of a crushable shell interacting with a rigid body with the CZone capability. The example represents a situation in which a thin part modeled with shells with initial velocity, $\boldsymbol{v_o}$, is crushed against a fixed rigid body using the CZone capability. During active crushing, the magnitude of the contact pressure is governed by the crush stress of the underlying material. In this example the magnitude of the contact pressure (shown in blue in *Figure 3*) does not change while crushing is active.

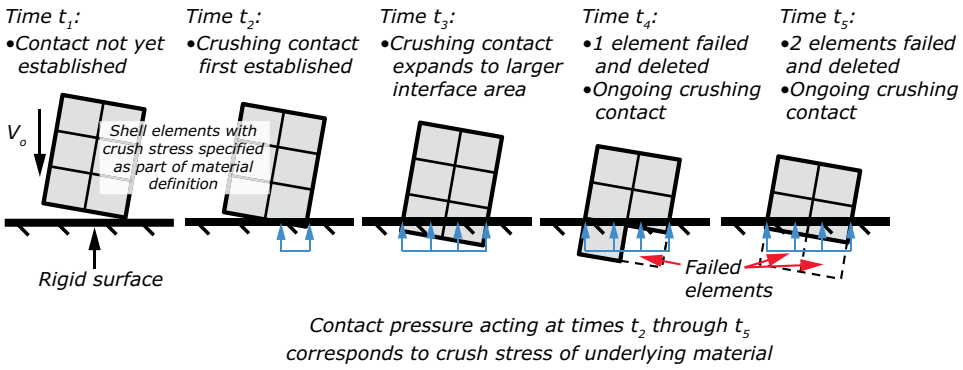


Figure 3: CZone continuous crushing sequence.

Significant penetrations can occur as a result of limiting the contact pressure to the crush stress of the underlying material. Shell elements involved in CZone crushing fail and are deleted once all nodes of a shell element penetrate the surface being impacted, such as in the configuration on the right side of *Figure 3*.

The crush stress can account for various material failure mechanisms, such as delamination, matrix cracking, fiber breakage, fiber pull-out, and fiber buckling, although the CZone capability does not attempt to distinguish between the various failure mechanisms. Even though the crush stress is specified as a material characteristic, the direct effect of the crush stress is limited to the contact pressure calculations; that is, constitutive (material) calculations for elements that are still active *do not* consider crush stress aspects of the material definition.

As discussed in the context of *Figure 3*, penetrations on the order of element dimensions are expected in the crushing region. The implementation of the CZone capability in general contact includes a shielding mechanism to avoid undesired secondary contacts for parts undergoing crushing. For example, this shielding mechanism avoids undesired interactions between the crushable shell and the body below the thin rigid body in *Figure 4*.

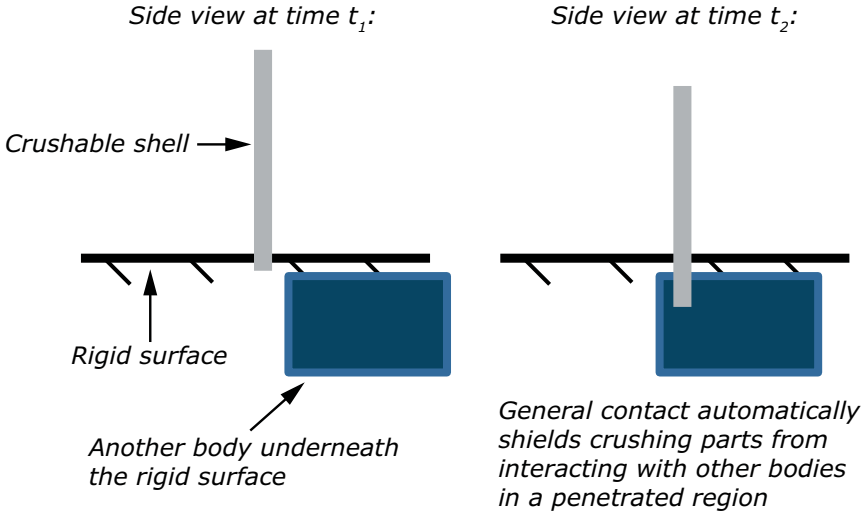


Figure 4: Example of automatic shielding from secondary contacts during CZone crushing.

Overview of Model Set-Up Aspects of CZone

Model specification aspects associated with CZone include:

- Modeling potentially crushing parts with shell elements (see *Shell Elements*);
- Including surfaces involved in crushing contact in the general contact definition (see *About General Contact in Abaqus/Explicit*);

- Specifying crush stress characteristics in the material definition associated with potentially crushing parts (see *CZone Crush Stress*);
- Specifying composite section and material characteristics other than crush properties. For example, composite layers can be modeled with anisotropic linear elastic material (see *Defining Orthotropic Elasticity in Plane Stress*) or with viscoelastic response (see *Time Domain Viscoelasticity*). Other failure mechanisms can be considered together with CZone crushing (see *Progressive Damage and Failure*);
- Optionally, specifying nondefault crush initiation characteristics as surface properties (see *Crush State Evolution*); and
- Optionally, requesting CZone output.

CZone Crush Stress

The crush stress can be thought of as the stress of the crushing body at the crushing interface. The crush stress is specified as a material property (see *CZone Crush Stress*) but actually does not directly influence the constitutive calculations associated with element integration points; rather, the crush stress limits the contact pressure magnitude at an actively crushing interface.

Contact Limitations for CZone for Abaqus

The following limitations currently exist for CZone for Abaqus:

- Crushable bodies can be modeled only with shell elements (see *Shell Elements*).
- Only node-to-surface contact is supported. Edge-to-edge contact is automatically deactivated for nodes involved in crushing; however, edge-to-edge contact may be active for nodes on crushable bodies that are not crushing.
- Crushing calculations cannot involve analytical rigid surfaces.

Complex Intersections and Domain Decomposition in CZone for Abaqus

Figure 5 shows a close-up view of a part with a T-junction geometry crushing against different sides of a folded shell in different regions. The folded shell experiences self-contact. This example shows that crushable bodies can have complex geometries, that crushing can occur with a double-sided target surface, and that noncrushing contact can occur nearby crushing contact.

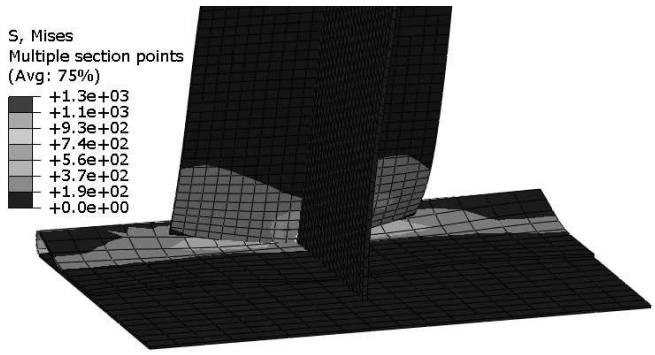


Figure 5: Crushing of a T-junction part against a folded shell surface.

The implementation of CZone for Abaqus allows the crushing region to be split across domains for domain decomposition associated with distributed memory parallelization.

Crush State Evolution

Table 1 summarizes the possible crush state settings for secondary nodes. The values in the first column of *Table 1* are reflected in the output variable CRUSHSTATE. Nodes can have initial states of "not crushable" (0), "non-trigger" (1), or "trigger" (2). "Not crushable" nodes have no crush associated with adjacent elements and will never crush. "Non-trigger" nodes will be reassigned "trigger" status during the simulation if adjacent nodes begin crushing or adjacent elements experience material failure.

Upon coming into contact, a "trigger" node develops contact pressures consistent with the assigned contact pressure-overclosure relationship (see *Contact Pressure-Overclosure Relationships*) and becomes an "actively crushing" node if either of the following conditions are met:

- the contact pressure acting in the plane of the crushable material underlying the secondary surface reaches the crush initiation stress, and the angle between contacting surfaces is greater than the crush initiation angle; or
- an adjacent secondary node is actively crushing, and the angle between contacting surfaces is greater than the crush continuation angle.

By default, the crush initiation stress is equal to the crush stress, but you can specify a scale factor to increase the crush initiation stress above the default value. Default thresholds of crush initiation and crush continuation are 30° and 25°, respectively. The angle between contacting surfaces is always in the range of 0° to 90°, so resetting either threshold angle to 0° results in all contacts meeting the respective angle threshold.

Table 1: Crush state settings for secondary nodes.

Value of Output Variable CRUSHSTATE	Description	Behavior
0	Not crushable node	This node will never crush; no crush stress is associated with this node.
1	Non-trigger node	This node is not a trigger node and had no prior crushing.
2	Trigger node	This node is a trigger node and had no prior crushing.
3	Previously crushing node	This node was previously crushing.
4	N/A (this setting is not used)	
5	Actively crushing node	This node is actively crushing.
6	Failed node	This node has "failed" as a result of crushing or material failure.

Contact forces for nodes with "actively crushing" status are equal to the crush stress times the crush area. A secondary node continues crushing until:

- It comes out of contact or the angle between contacting surfaces falls below the crush continuation angle. In this case the nodal state is reset to "previously crushing."
- All adjacent elements are deleted due to crushing or other types of material failure. In this case the nodal state is "failed."

A node with a "previously crushing" status resumes crushing if the angle between contacting surfaces is greater than the crush continuation angle and either of the following conditions are met:

- the contact pressure acting in the plane of the crushable material underlying the secondary surface reaches the crush stress; or
- an adjacent secondary node is actively crushing.

Input File Usage: Use the following option to initialize a node to "trigger" status (default) and to specify a crush initiation scale factor, crush initiation angle, and crush continuation angle:

**SURFACE PROPERTY ASSIGNMENT*, PROPERTY=CRUSH TRIGGER
surface name, TRIGGER, *scale factor*, *crush initiation angle*, *crush continuation angle*

Use the following option to initialize a node to "non-trigger" status, and (in case the node later becomes a trigger node) to specify a crush initiation scale factor, crush initiation angle, and crush continuation angle:

**SURFACE PROPERTY ASSIGNMENT*, PROPERTY=CRUSH TRIGGER
surface name, NO TRIGGER, *scale factor*, *crush initiation angle*, *crush continuation angle*

Use the following option to initialize a node to "not crushable" (no crush) status:

**SURFACE PROPERTY ASSIGNMENT*, PROPERTY=CRUSH TRIGGER
surface name, NO CRUSH

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **General contact (Explicit)**: **Surface Properties**: **Crush Trigger assignments: Edit**:
Select surface or material, click the arrows to transfer surface or material to list of thickness assignments.

Influence of Noncrushing Failure Mechanisms on Crush Initialization

Failure mechanisms other than CZone crushing (see *About Progressive Damage and Failure*) can influence CZone crushing. For example, element failure due to a damage mechanism discussed in *Damage Evolution and Element Removal for Fiber-Reinforced Composites* will cause adjacent "non-trigger" nodes to become "trigger" nodes. Criteria for a "trigger" node to become "actively crushing" are discussed in *Crush State Evolution*.

Testing Crush Behavior

Until the micromechanics of crushing failure mechanisms are more fully understood, it is necessary to test the crush properties of candidate materials. There are several methods to test crush strength; some of these methods are still evolving. Once the crush strength is determined, the crush stress for the composite can be calculated.

This section discusses three methods used to test the crushing properties of composite materials. To obtain the best simulation results, the tests used to determine the crushing properties should represent the design structure as closely as possible.

Ultimately, you must decide which method of crush strength measurement to use in determining the crush stress values to input for the CZone for Abaqus analysis. Regardless of the method selected, the coupon design should have a thickness and layup consistent with the structure being developed.

Component Testing

In the component testing method a component is manufactured from the candidate material. The component geometry should support crushing, such as a straight box or similar section with a side length to wall thickness ratio that prevents local buckling. This is the most accurate testing method; however, it is also relatively costly. This approach involves the following steps:

1. Crush the component using an impact sled.
2. Measure the acceleration of the sled.
3. Calculate the crush force from the acceleration.

4. Calculate the crush stress by dividing the crush force by the cross-sectional area of the component.

The high cost of this test method has led to the development of lower cost “coupon” test methods described in the following sections.

Flat Coupon Testing

Several organizations are developing the flat coupon testing method. Test coupons are typically cut from a flat sheet using a water jet to minimize damage at the cut edges. One end of the coupon is weakened by incorporating a “saw-tooth” shape or chamfer to initiate crushing. The flat coupon is placed in a fixture that prevents buckling. A drop-tower or servo-hydraulic test apparatus is used to push the coupon through the fixture onto a crushing or target plate. The ongoing crush force is divided by the cross-sectional area of the flat coupon to determine the crush strength.

In the flat coupon testing method, the coupon typically has a region near the crushing end that is free from support or constraint. This region is the unsupported length. *Figure 6* shows a coupon crushing fixture installed in a servo-hydraulic dynamic testing apparatus.



Figure 6: Flat coupon test apparatus (courtesy of Engenuity, Ltd.).

Figure 7 shows how the material crushes at the lower end of the fixture as it is forced down through the fixture's low-friction guides. The fixture has adjustments for different coupon thicknesses and for varying the unsupported length.



Figure 7: Crushing a flat coupon (courtesy of Engenuity, Ltd.).

Shaped Coupon Testing

Several organizations are developing the shaped coupon testing method. The test coupons are shaped with corrugations to give them stability when they are crushed on a flat surface. Like the flat coupons, one end of the shaped coupon is weakened with a “saw-tooth” shape, as shown in *Figure 8*, or a chamfer to initiate crushing.



Figure 8: A typical shaped coupon used for crush testing (courtesy of Engenuity, Ltd.).

Figure 9 shows the shaped coupon after testing.



Figure 9: The crushed shaped coupon (courtesy of Engenuity, Ltd.).

The shaped coupons are more expensive than the flat coupons but still significantly less expensive than producing custom test components.

Comparison of Flat and Shaped Coupon Tests

Many composites exhibit ongoing delamination in the crush zone as part of the crushing process. Shaped coupons tend to suppress delamination, often leading to a higher value for the crush stress than the flat coupons, which allow delamination more readily. The ratio of the measured crush stresses from the shaped and flat coupon methods varies with the material.

Most composite structures consist of both flat and curved regions. To evaluate a realistic crush performance for these structures, it is recommended to test samples using both coupon methods. You can then assign the flat coupon crush stress values to flat regions and the shaped (suppressed delamination) crush stress values to curved regions. This approach is illustrated in *Figure 10*.

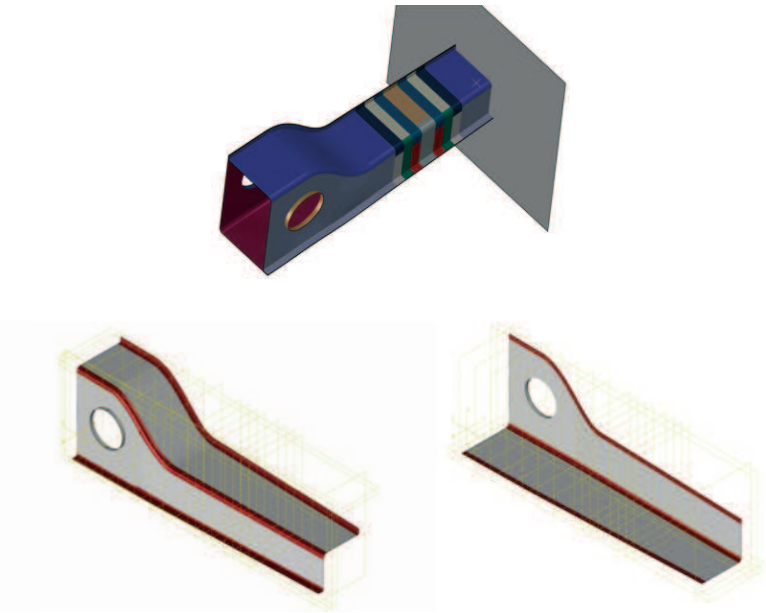


Figure 10: Identifying regions of higher crush resistance based on curvature.

The top image shows a CZone for Abaqus analysis of a cone made from two components joined at the flanges. The bottom images show the regions of crush stress assignment in the two pieces that make the cone. The corner radius areas, including the flange areas where the two pieces are joined, help to suppress delamination. They are assigned a delamination suppressed crush stress of 122 MPa. The flat areas are assigned a flat coupon crush stress of 50 MPa. The physical test of the component yielded an average crush stress of 77 MPa.

Some materials have a high resistance to delamination, such that the flat and shaped coupon test methods converge toward a common crush stress value. Further research is being conducted into methods of suppressing delamination in the flat coupon test method to determine the elevated crush strength values while using the low cost flat coupons.

Output

Viewing the results of a CZone for Abaqus analysis is comparable to viewing the results of other Abaqus analyses. To understand the results of a crushing analysis, you should review the output results associated with the CZone crushing state, the material damage, and the energy quantities in the Visualization module of Abaqus/CAE. For general information on viewing results, see *Viewing results*.

CZone Crushing State and Material Damage Variable Output

CZone for Abaqus uses the contact output variable CRUSHSTATE to report information about the current crushing state of the nodes on the secondary surface (see *Table 1*).

In addition, you should request output of the relevant damage variables in accordance with the composite damage model used in the simulation. For example, for the Tsai-Wu criterion you need output of solution-dependent state variables, and for the Hashin damage model you need output of the damage variables documented in *Output*. You can use these results to determine the regions that failed due to crushing and those that failed as a result of composite damage and failure.

Highlighting Failed Regions

Plot the contours of CRUSHSTATE on the undeformed shape to highlight the regions that have failed or are starting to fail as a result of crushing. The regions where CRUSHSTATE is six (colored red in *Figure 11*) have failed as a result of crushing. Regions where CRUSHSTATE is five (colored yellow) are currently crushing.

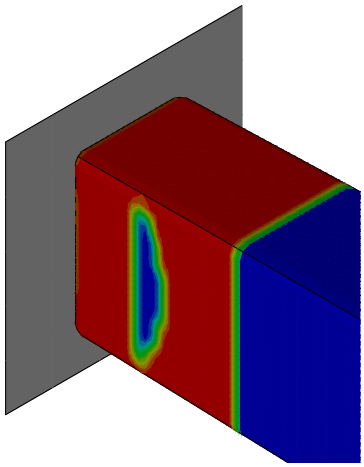


Figure 11: Plotting output variable CRUSHSTATE on the undeformed shape.

Removing Failed Elements from the View

You can use **Result->Field Output->Status Variable** to remove the failed elements from the view. Select CZASTATE as the output variable, and remove elements with a value greater than 3; select **Apply to undeformed state** to remove the elements from the undeformed model view. The result is shown in *Figure 12*.

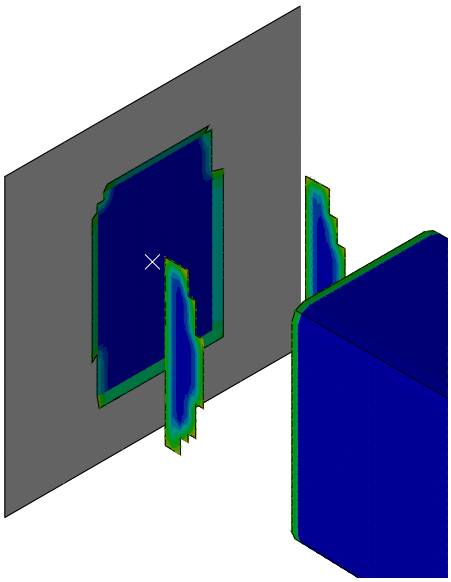


Figure 12: Fragments that broke away from the model as the surrounding elements failed.

Removing Elements That Failed due to Material Damage

If you use the Tsai-Wu damage initiation model, you can remove the elements that have failed as a result of material damage. Select `SDV3` to be the status variable, and remove elements whose `SDV3` value is greater than 0.99, as shown in *Figure 13*.

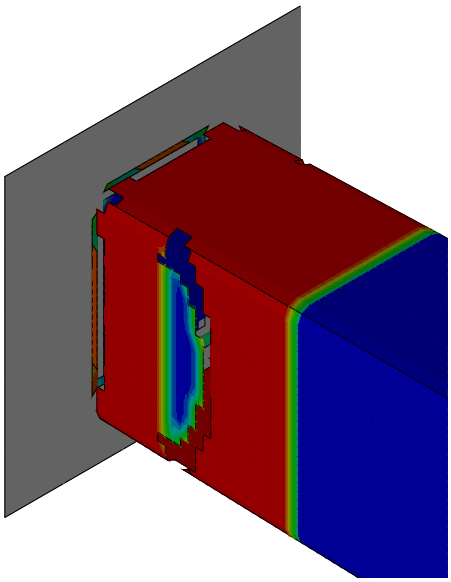


Figure 13: Removing elements that have failed as a result of material damage.

If you are using the Hashin damage model, you can request the damage output variables associated with different damage mechanisms to evaluate the dominant damage mode (see *Output*).

Example Crushing Simulations

This section contains two benchmark simulations that you can use as guides for creating new analyses. An input file for each benchmark problem can be extracted as described in *Fetching Sample Input Files*.

Plain Cone

In this benchmark a rectangular cross-section cone is attached to a rigid wall, as shown in *Figure 14*, and impacted with a moving sled with a mass of 324 kg and an initial velocity of 10m/s. The input file for this example is `cza_bm1-cone-ahv_gc.inp`.

The layup of the composite cone includes three different thickness regions. It is thinnest at the front (left side of the plain cone test setup) and gets thicker toward the rear (right side of the plain cone test setup), as shown in *Figure 14*. The thickness at the front of the cone is such that local buckling occurs and fragments of material break away due to bending stresses. As crushing progresses, the increased wall thickness allows progressive crushing of the cone without buckling and bending failure. *Figure 14* shows the final crushed state of the cone.

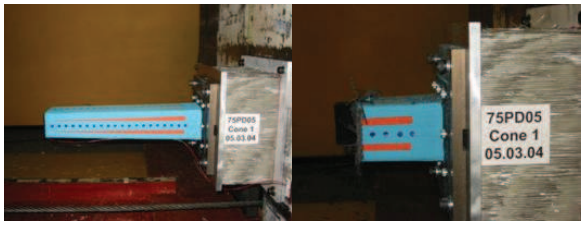


Figure 14: The plain cone test setup (left) and the plain cone sample at test completion (right) (courtesy of Engenuity, Ltd.).

Figure 15 shows some material fragments that broke away from the front of the cone.

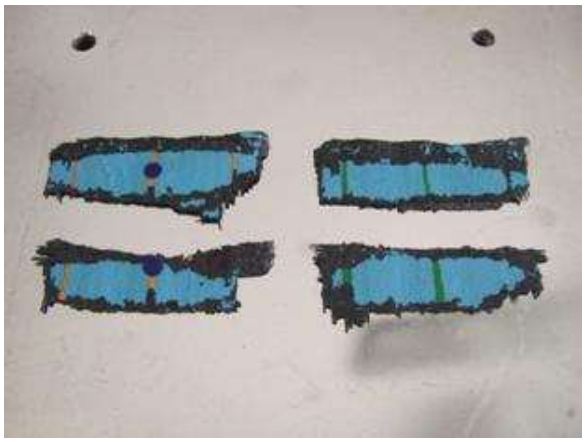


Figure 15: Fragments taken from the experimental test (courtesy of Engenuity, Ltd.).

The tapered colored markings on the fragments indicate that they came from the front of the cone on the long sides. The fragments that were predicted to break out from the thin part of the cone in the simulation match well with the test fragments, as shown in *Figure 16*.

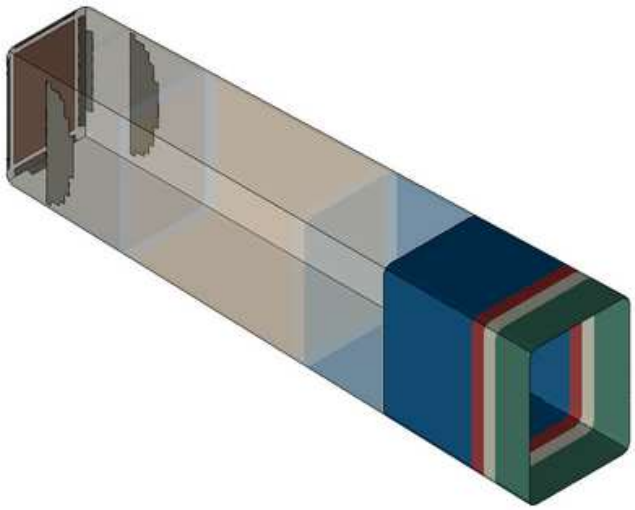


Figure 16: Fragments that broke away during the simulation.

The predicted sled acceleration curve also matches well with the test data, as shown in *Figure 17*.

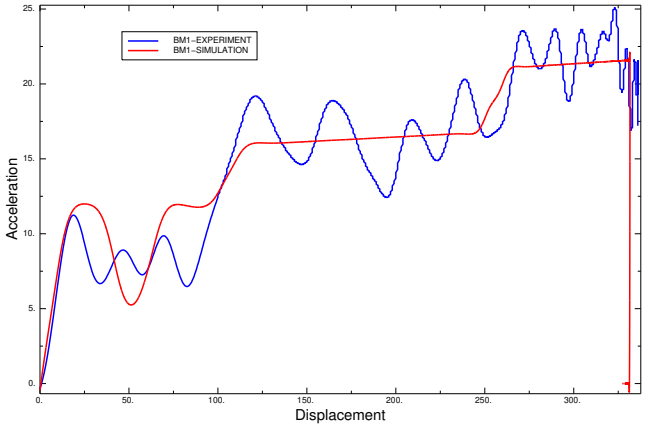


Figure 17: History of the sled acceleration for the simulation and experiment.

Complex Cone

In this benchmark a complex cone is attached to a moving sled, as shown in *Figure 18*, and impacted against a wall. The input file for this example is `cza_bm3-b-cone-ahv_gc.inp`. The sled has a mass of 1150 kg and an initial velocity of 9.1 m/s.

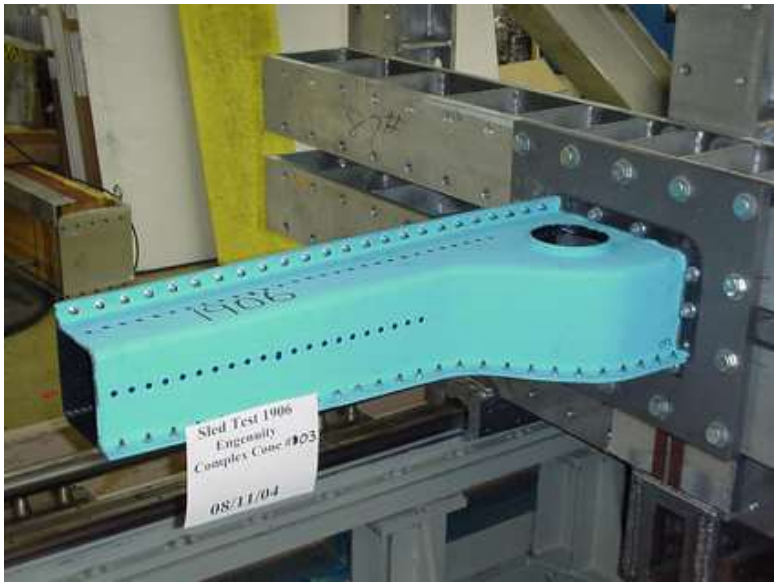


Figure 18: Experimental setup for the complex cone (courtesy of Engenuity, Ltd.).

The component geometry is intended to represent some of the typical features and general size of a typical automotive front longitudinal member. The layup of this component was designed to behave in a way that shows an initial period of crushing, followed by catastrophic failure of the backup structure. At the front of the component the layup is relatively thin, and the backup structure is strong enough to support the crushing forces. As crushing progresses, the thickness of the layup progressively increases. At a certain point the back of the cone can no longer support the increased crushing forces from the thicker material, and failure occurs at the curved transition area—this quickly propagates around the section and the cone breaks in two. *Figure 19* and *Figure 20* show experimental and analysis results, respectively, at the point of failure.



Figure 19: Experimental results for the complex cone (courtesy of Engenuity, Ltd.).

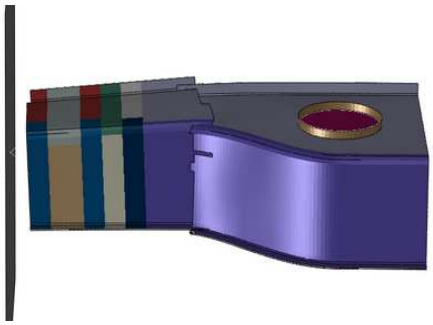


Figure 20: Simulated failure of the complex cone.

Acceleration histories of the experimental and analysis results correlate well, as shown in *Figure 21*. The acceleration of the analysis approaches a constant value near zero after the crushing event, as shown by the red curve. The acceleration of the experimental results exhibit some experimental noise after passing through zero, as shown by the blue curve.

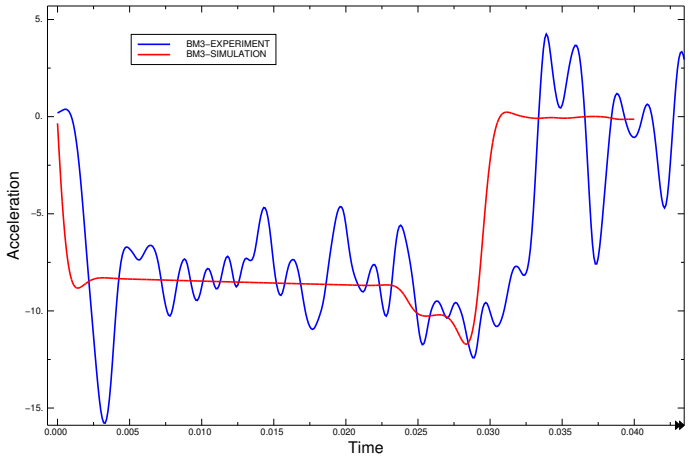


Figure 21: Time history of the sled acceleration for the complex cone.

Verification Tests

This section contains a series of verification tests for CZone for Abaqus. These simple problems are intended to demonstrate some of the capabilities of the CZone methodology used with an Abaqus/Explicit analysis and to allow you to confirm the results. An input file for each verification problem can be extracted as described in *Fetching Sample Input Files*.

Single Strip

This first, simple example tests the failure of elements and the correct calculation of the crushing forces through the CZone for Abaqus interaction. A sled, modeled as a discrete rigid body, impacts a simple composite strip. The composite strip has an encastre boundary condition at one end and is constrained to remain planar. The acceleration versus time history should be constant once the crushing mode is established. The crushing stress for the material is independent of the direction in which it is being crushed.

The results are obtained by running `cza_v1_gc.inp`. The time history of the sled acceleration (history variable `Spatial acceleration: A1 PI: SLED-1 Node 37 in NSET SLED-RP`) should be plotted to confirm the constant loading, as shown in *Figure 22*.

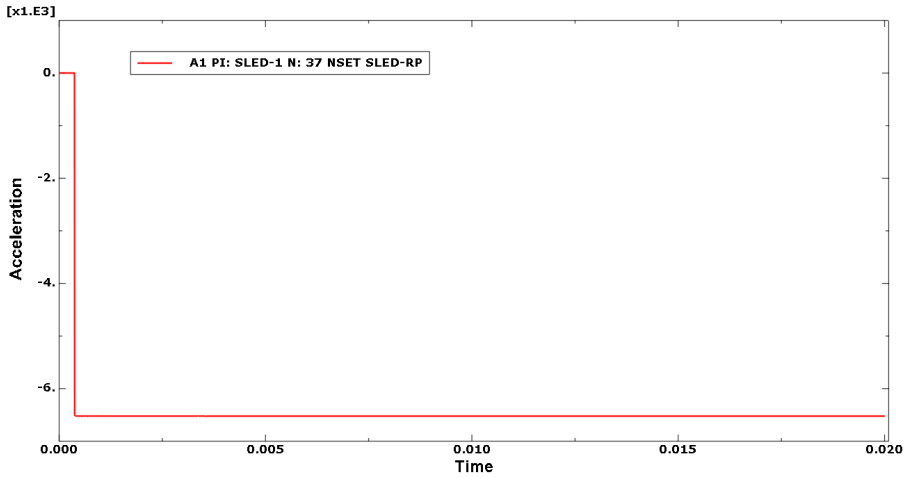


Figure 22: The time history of acceleration for the single strip.

Angled Strip

In this test the plate is angled relative to the strip of composite material. The crushing force should gradually increase as the sled makes contact with the strip before the crushing force becomes approximately constant. The crushing stress for the material is independent of the direction in which it is being crushed.

The results are obtained by running `cza_v2_gc.inp`. The time history of the sled acceleration (history variable `Spatial acceleration: A1 PI: SLED-1 Node 37 in NSET SLED-RP`) should be plotted to confirm the loading, as shown in *Figure 23*.

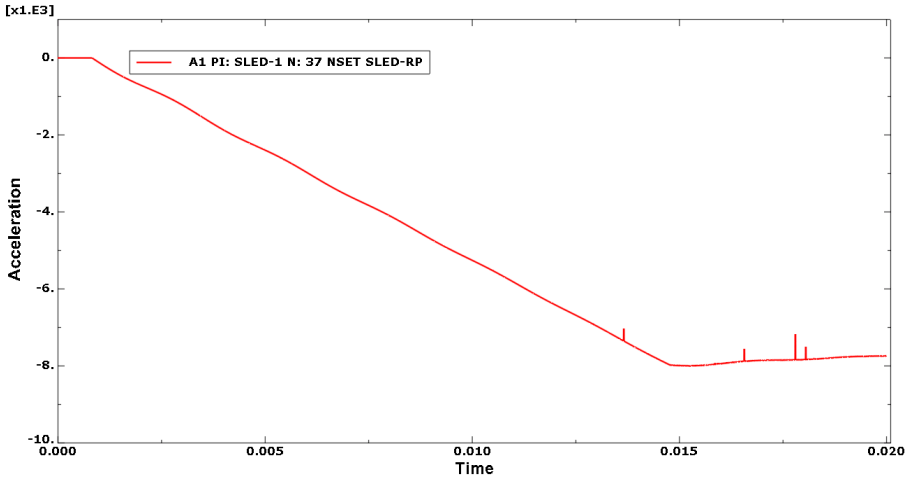


Figure 23: The time history of acceleration for the angled strip.

Bounce (Oblique Impact)

CZone for Abaqus contact definitions are described in *CZone Methodology*. This example tests an oblique (non-crushing) impact. The input file for this example is `cza_v3_gc.inp`. A composite strip is moving toward an oblique rigid plate, which is constrained in all degrees of freedom. The composite structure should hit and bounce clear from the plate. The filtered time history of the force on the rigid reference node of the plate is shown in *Figure 24*.

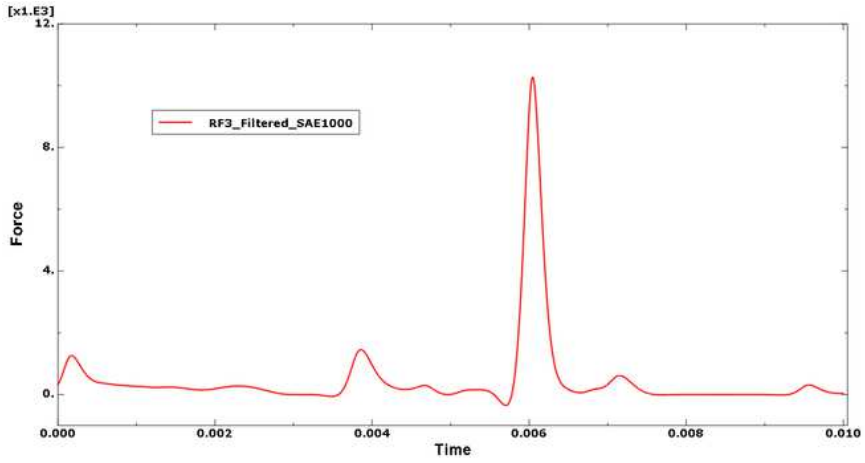


Figure 24: The time history of the force on the rigid reference node.

Friction

A very simple test is performed to check the friction calculations. The input file for this example is `cza_v4_gc.inp`. Two parts are defined, each containing a single element, but one part larger than the other. The larger part is a rigid surface, encastre at the rigid reference node, and the other is a very simple composite layup with two plies. The two parts are pressed together by a distributed load acting on the composite. Relative motion between the surfaces is induced by boundary conditions acting at the corner points of the composite element. The effective coefficient of friction can be calculated by monitoring the reaction force in the 1- and 2-directions and dividing it by the reaction force in the 3-direction. *Figure 25* and *Figure 26* show the friction forces in the 1- and 2-directions, respectively. The friction coefficient is shown in *Figure 27*.

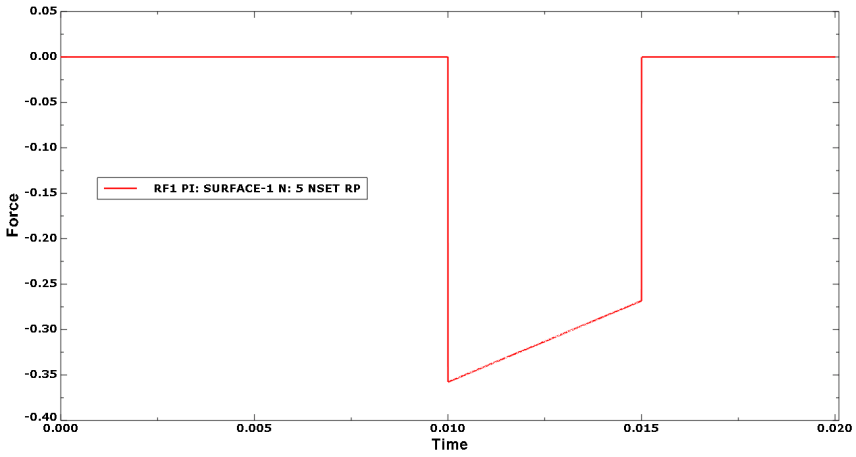


Figure 25: The time history of friction force in the 1-direction.

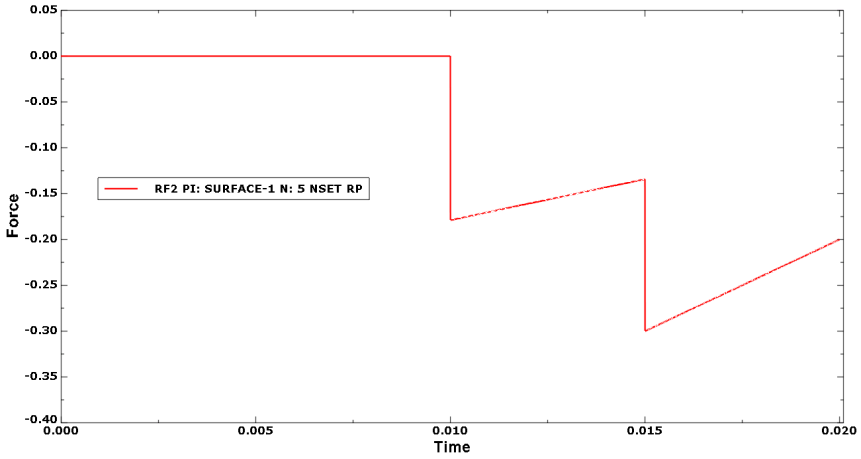


Figure 26: The time history of friction force in the 2-direction.

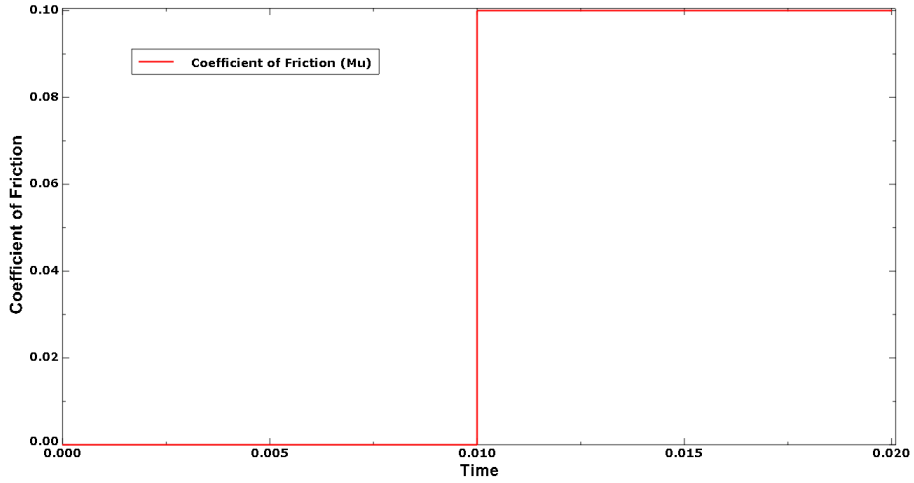


Figure 27: Time history of the friction coefficient.

Five Finger

In this test a single composite specimen, with five fingers and a central palm, is completely constrained. The composite has a simple layup composed of 10 layers of orthotropic material. The crush resistance is a function of angle for this material defined for the crush stress. The input file for this example is `cza_v5_gc.inp`.

Five individual sleds are defined, each paired with a finger on the composite component, as shown in *Figure 28*.

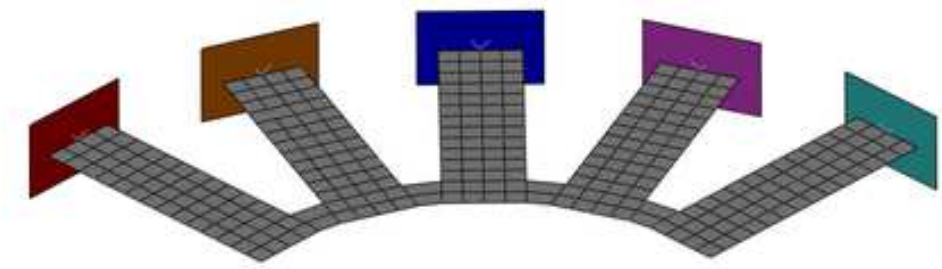


Figure 28: The initial configuration of the five finger model.

The sleds travel toward and along the length of the fingers with an initial velocity. The progress of the sleds is resisted by the force required to crush the composite fingers. The angular dependence of the crush resistance means that each sled travels a different distance along the fingers. The displacement versus acceleration for each sled is shown in *Figure 29*.

Note:

Two of the curves—the ones for nodes 1290 and 1292—are identical, so there appear to be only four curves in the graph. This is the result of the angular properties of the second and fourth fingers having identical crushing properties in the test direction.

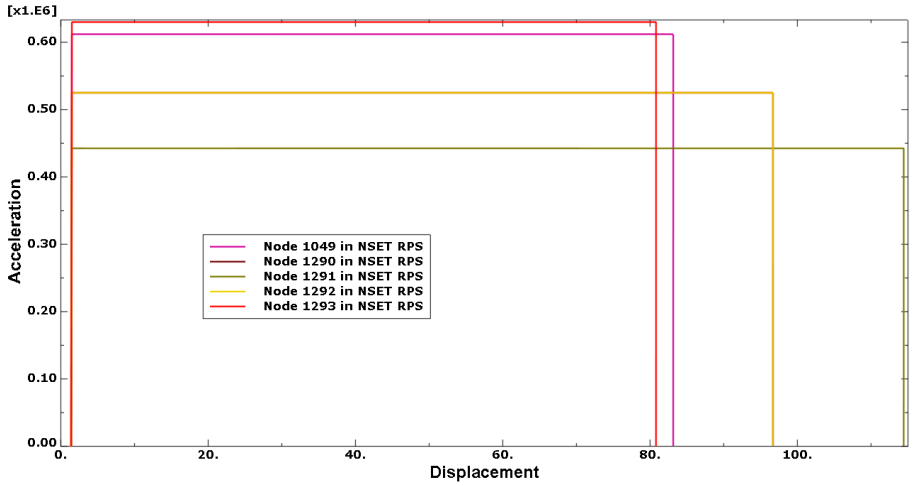


Figure 29: The displacement versus acceleration for each of the five sleds.

Velocity Dependence

This test uses the same composite geometry and layup of the *Five Finger* test. In this case the crush resistance is a function of crush velocity instead of angle. The crush stress velocity factor defines a scaling factor that is applied to the crush resistance based on velocity. The input file for this example is `cza_v6_gc.inp`.

Five individual sleds are defined, each paired with a finger on the composite component, as shown in *Figure 28*. Each sled travels with a different initial velocity defined along the length of the finger. The progress of the sleds is resisted by the force required to crush the composite fingers. The velocity-dependent crush properties and the different initial velocities generate a different crushing resistance and, therefore, a different acceleration, for each sled (see *Figure 30*).

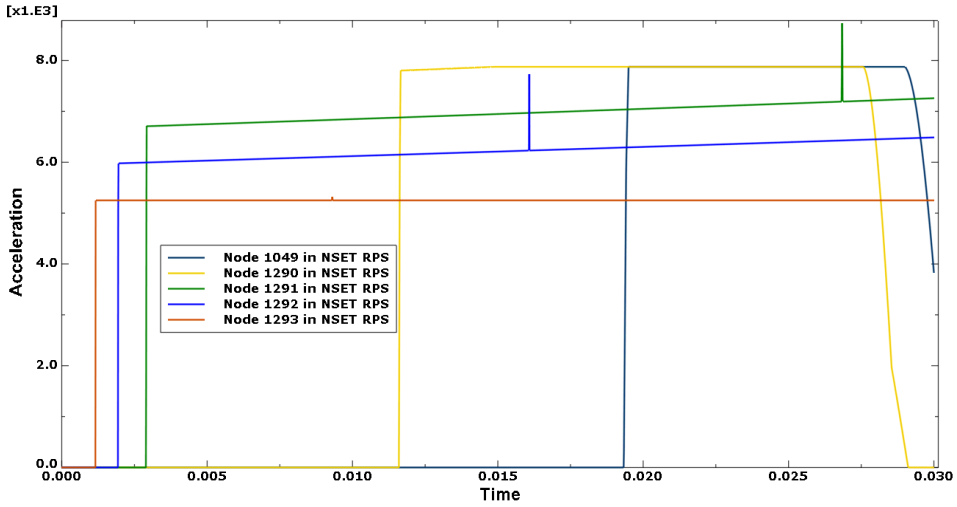


Figure 30: The time history of acceleration for the five sleds with different initial velocities.

Oblique Box (General Contact, Fixed Wall, Moving Cone)

The geometry and layup of the plain cone benchmark is reused in this verification problem. For information about the model, see *Plain Cone*. The cone is rotated relative to the sled, and the sled is constrained to remain stationary. An initial velocity of 10 m/s is applied to the cone and to the 500 kg mass attached to the back of the composite structure. The input file for this example is `cza_v7_gc.inp`.

A combination of crushing and brittle failure is predicted by the model. As the front face of the cone makes contact with the rigid surface, it exhibits brittle failure, peeling along the edges that join it to the side walls. This exposes new edges to a crushing interaction with the rigid surface. The mixture of brittle failure and crushing produces a noisy reaction force at the rigid reference node.

Leading Edge (Airfoil/F1 Wing)

Here, a simplified airfoil representing the nose cone of a Formula One car makes contact with a rigid pole. The input file for this example is `cza_v8_gc.inp`. The car is traveling at approximately 140 km/h. The initial impact causes brittle failure on the leading edge as the composite makes contact with the cone. The model setup is shown in *Figure 31* and *Figure 32*.

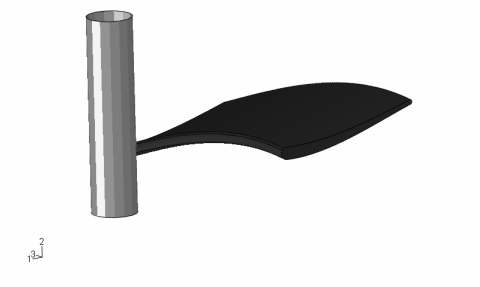


Figure 31: Airfoil contact, side view.

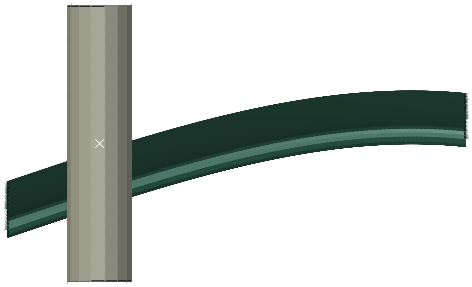


Figure 32: Airfoil contact, front view.

After initial impact, the failure mode becomes crushing as the impact continues and new composite material is exposed. Eventually the forces become too great, and failure is observed where the airfoil joins the nose (the boundary condition on the edge of the airfoil away from the pole). As this failure occurs, the composite structure rotates and the crushing ceases. This verification problem tests the implementation of the Tsai-Wu failure criteria and the transition from bouncing (hard contact) to crushing.

The difference in the failure modes is shown in *Figure 33* and *Figure 34*. All failed elements from both crushing and brittle failure have been removed from *Figure 33*. Only elements that failed as a result of crushing have been removed from *Figure 34*. The removal of failed elements in Abaqus/CAEis discussed in *Output*.

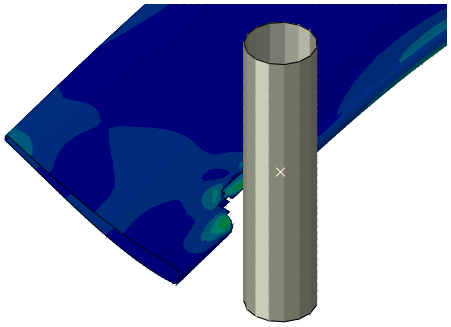


Figure 33: Airfoil with all failed elements removed.

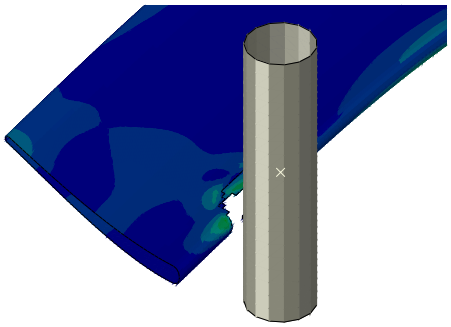


Figure 34: Airfoil with only crushed elements removed.

Composite Failure, Part One

A tapered composite sample is crushed against a stationary rigid surface, as shown in *Figure 35*. The input file for this example is `cza_v9_gc.inp`.

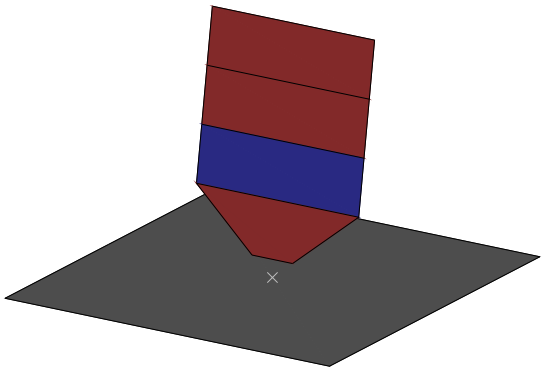


Figure 35: The composite failure model setup.

The crush force increases as the first tapered element crushes against the rigid surface. The increasing load causes the element behind the tapered element to fail as a result of brittle failure. The first element is free to bounce from the surface as the rest of the structure continues toward it. The reaction force at the rigid reference node is shown in *Figure 36*.

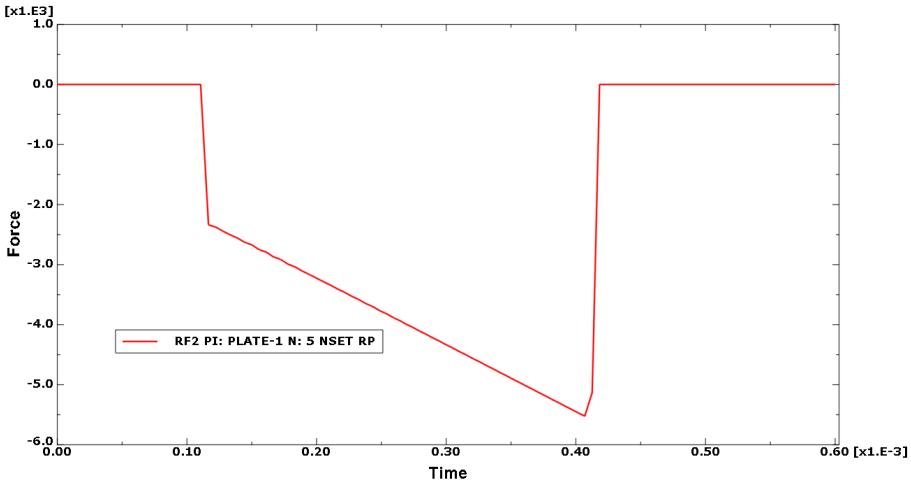


Figure 36: The reaction force at the rigid reference node.

Composite Failure, Part Two

A coarse representation of a cylindrical component is used to test the damage and failure of elements close to the crush zone interaction. The input file for this example is `cza_v10_gc.inp`. The component, shown in *Figure 37*, is pushed against a flat, rigid, encastre surface.

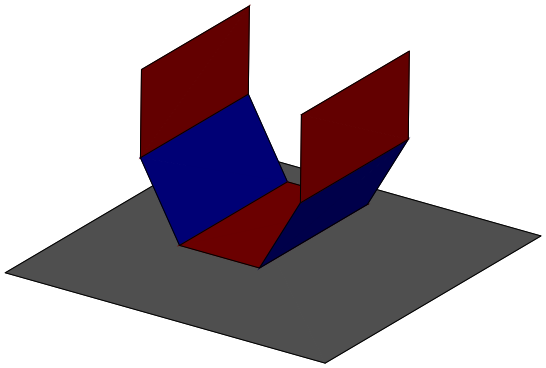


Figure 37: The composite failure model setup.

The first element bounces against the rigid surface and transfers the load through the neighboring elements to the rest of the structure. Eventually the two neighboring elements fail as a result of damage mechanisms. This releases the central element, which bounces from the surface. The component continues to move toward the rigid surface, and eventually the side walls begin to crush. The time history of the reaction forces is shown in *Figure 38*.

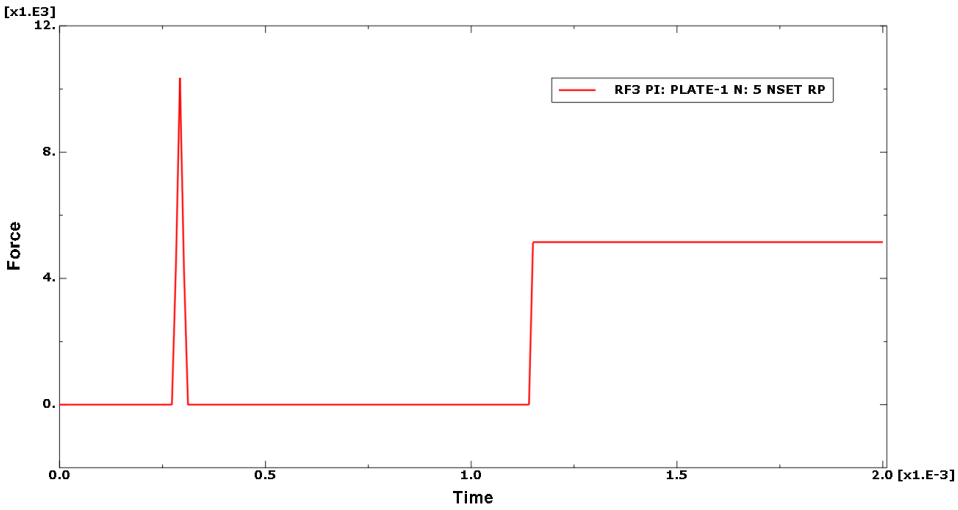


Figure 38: Reaction forces during the composite failure.

Input File Template

The following template includes the basic Abaqus options and all of the available CZone for Abaqus options required to create a crushing analysis. There are two blocks in the template, each starting with a material definition. The first block is for use with Tsai-Wu damage initiation. The second block is for use with Hashin damage initiation. The rest of the template continues with all material references made to the Tsai-Wu model. The template uses the parts and instances input file format used in Abaqus/CAE; however, CZone for Abaqus does not require that you use this format.

```
*HEADING
...

*PART, NAME=Part-1
Node, element, section, set, and surface definitions
```

```
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
…
*END ASSEMBLY

**
**Material with Tsai-Wu damage
**
*MATERIAL, NAME=material1
*PARAMETER
Independent parameter definitions for plane stress orthotropic elasticity
*ELASTIC, TYPE=LAMINA
Data lines to define the material constants
Data lines to define the material constants with damage evolution
*DEPVAR, DELETE=1
Data lines to define the 3 dependent variables (status, Tsai-Wu, and stiffness degradation)
*USER DEFINED FIELD, CZONE, PROPERTIES=16
Data lines to define Tsai-Wu failure stresses and the original material stiffness and absorbed energy
*CRUSH STRESS
Data lines to define the crush stress as a function of orientation
*CRUSH STRESS VELOCITY FACTOR
Data line to define the stress scaling factor as a function of relative velocity (optional)
**
**End Tsai-Wu material definition
**

**
**Material with Hashin damage
**

*MATERIAL, NAME=material2
*PARAMETER
Independent parameter definitions for plane stress orthotropic elasticity
*ELASTIC, TYPE=LAMINA
Data lines to define the material constants
*DAMAGE INITIATION, CRITERION=HASHIN
Data lines to define the Hashin failure criterion
*DAMAGE EVOLUTION
Data lines to define the evolution of damage leading to eventual failure
*CRUSH STRESS
Data lines to define the crush stress as a function of orientation
*CRUSH STRESS VELOCITY FACTOR
Data line to define the stress scaling factor as a function of relative velocity (optional)
**
**End Hashin material definition
**


*SURFACE INTERACTION, NAME=Int-1
*FRICTION
Specify the friction coefficient

*INITIAL CONDITIONS, TYPE=VELOCITY
Nodeset1, dof, initial_velocity
```

```
**
*STEP, NAME=Step-1
*DYNAMIC, Explicit
, time
**
**Optional boundary condition for use with *CRUSH STRESS VELOCITY FACTOR
**
*BOUNDARY, TYPE=VELOCITY
Data line to define stress scaling factor and relative velocity
*CONTACT
*CONTACT INCLUSIONS, ALL EXTERIOR
*CONTACT PROPERTY ASSIGNMENT
 , , Int-1

**
**Output requests
**
*RESTART, WRITE, NUMBER INTERVAL=integer, TIME MARKS=NO
*OUTPUT, FIELD, VARIABLE=PRESELECT
...
Specify desired element, contact, and history output requests
*END STEP
```

Additive Manufacturing Process Simulation

Additive manufacturing offers the ability to produce complex designs not possible to create with traditional manufacturing methods.

In this section:

- *About Additive Manufacturing Process Simulation*
- *Toolpath-Mesh Intersection Module*
- *Thermomechanical Simulation of Additive Manufacturing Processes*
- *Eigenstrain-Based Simulation of Additive Manufacturing Processes*
- *Special-Purpose Techniques for Additive Manufacturing*

About Additive Manufacturing Process Simulation

Products: Abaqus/Standard

References:

- *Toolpath-Mesh Intersection Module*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

Additive manufacturing (AM) offers the potential to produce complex parts that are not possible to produce using traditional manufacturing methods.

This section describes Abaqus/Standard functionality intended for the simulation and evaluation of additive manufacturing processes and the impact process parameters have on the final printed part.

References:

- *Toolpath-Mesh Intersection Module*
- *Special-Purpose Techniques for Additive Manufacturing*

About Additive Manufacturing

Additive manufacturing, also referred to as 3D printing, is a broadly used term to describe industrial processes by which three-dimensional objects are manufactured through:

- a controlled deposition of raw material (typically in powdered, melted, or liquid state); and
- induced transformation into a solid state.

Common additive manufacturing processes (ISO / ASTM52900-15) are described in the table below.

Technique	Powder bed	Binder jetting	Directed energy deposition	Material extrusion	Sheet lamination	Photo polymerization	Material jetting
Description	Thermal energy selectively fuses regions of a powder bed.	A liquid bonding agent is deposited to join powder materials.	A nozzle mounted on a multi-axis arm deposits melted material.	Material is drawn via a nozzle, where it is heated. It is deposited layer by layer.	Sheets of material are bonded to form an object.	Liquid photopolymer is selectively cured by light-activated polymerization.	Droplets of build materials are selectively deposited.
Material Form	Powder	Powder	Powder or wire	"Solid" material	"Solid" material	Liquid resin	Ink
Material	Metal Plastic	Metal Plastic Ceramics	Metal	Plastic Composite	Paper Metal	Plastic (photopolymer resin)	Plastic

Processes Terms	Selective laser sintering (SLS) Selective laser melting (SLM) Electron beam melting (EBM) Direct metal laser sintering (DMLS)	Binder jetting (BJ) Inkjet powder printing Multi jet fusion (MJF)	Laser cladding Direct energy deposition (DED) Laser metal deposition (LMD) Laser engineered net shape (LENS) Laser or electron beam wire deposition	Fusion deposition modeling (FDM)	Laminated object manufacturing (LOM) Paper lamination technology (PLT) Ultrasonic additive manufacturing (UAM)	Stereo lithography (SLA) Digital light processing (DLP)	Photopolymer jetting (PolyJet) Multi jet modeling (MJM)

Additive manufacturing makes it possible to produce complex shapes not subject to the design constraints of more traditional manufacturing methods. Therefore, the functional requirements of a part become the primary focus of the design effort. However, additive manufacturing processes have their own challenges. For example, thermal strains induced by the manufacturing process can produce residual stresses large enough to cause failure during printing or during the in-service life of a part.

Some of the main objectives of an additive manufacturing simulation are to:

- Predict the residual stresses in a part.
- Minimize the gap between the designed and manufactured part through process optimization.
- Evaluate how a manufactured part performs under realistic loading conditions in an assembly with other components.

At the core of the Abaqus/Standard additive manufacturing technology is the toolpath-mesh intersection module—a powerful geometry-based engine that takes process toolpath data as input and intersects it with an arbitrary mesh.

Abaqus/Standard provides two methods for the simulation of an additive manufacturing process: a thermomechanical simulation and an eigenstrain-based simulation.

Abaqus/Standard offers general-purpose simulation capabilities that allow you to define appropriate boundary conditions, loads, interactions, constraints, and material models required to capture the physics of additive manufacturing processes. In addition, special analysis techniques are available for the simulation of additive manufacturing processes that take into account machine information and process parameters, such as laser power, layer thickness, and toolpath. Abaqus/Standard also allows you to perform postprocessing simulations and in-service performance validations for printed parts.

Toolpath-Mesh Intersection Module

The toolpath-mesh intersection module is used to find geometric intersections between various toolpaths used in additive manufacturing processes and a finite element mesh of the part to be manufactured. For example, in a typical metal powder bed simulation, you can define the time-location history of both powdered metal deposition and a heating source (such as a laser) using an event series. The toolpath-mesh intersection module uses the event series data and automatically computes the relevant information required to activate elements and to apply the proper thermal energy to the model. For more information, see *Toolpath-Mesh Intersection Module*.

Thermomechanical Simulation

A thermomechanical simulation consists of a transient heat transfer analysis of thermal loads introduced on a part during the printing process followed by a static structural analysis that is driven by the temperature field from the thermal analysis. The simulation allows exact specification in time and space of processing conditions and gives you precise control over the fidelity of the solution. The simulation is accurate and comprehensive,

but it can be computationally expensive with increasing time and spatial (mesh) resolution. For more information, see *Thermomechanical Simulation of Additive Manufacturing Processes*.

Eigenstrain-Based Simulation

Eigenstrain, or inherent strain, is an engineering concept used to account for all possible sources of permanent (or inelastic) deformation induced by the process. Eigenstrain has long been used for the evaluation of residual stresses from welding operations. An eigenstrain-based simulation of an additive manufacturing process consists of a single static stress analysis of a printing part where a predefined eigenstrain field is applied according to an element activation sequence representative of the process (usually layer by layer). This process results in a distribution of residual stresses and a deformation field that can lead to the overall distortion of the part. This method eliminates the need to obtain detailed machine information; however, it requires additional effort to calibrate eigenstrain values from experiments or detailed process-level thermomechanical simulations. The results from an eigenstrain-based simulation are typically more approximate than a thermomechanical simulation. Generally, the eigenstrain analysis is sufficient to capture distortion and residual stresses but may not capture higher-order deformation modes, such as the buckling that can happen in thin-walled components during printing. For more information, see *Eigenstrain-Based Simulation of Additive Manufacturing Processes*.

Special-Purpose Techniques for Additive Manufacturing

The functionality in Abaqus/Standard for additive manufacturing processes simulation is developed on a user subroutine infrastructure and keyword interface that provides a high degree of control and customization. In addition, a number of special-purpose techniques are available for simulation of common AM processes that do not require you to write user subroutines. These techniques are implemented as "internal" user subroutines in Abaqus using the same user subroutine infrastructure and keyword interface. These special-purpose techniques are accessed by using table collections with string names starting with **"**`ABQ_`**"** as is described in detail in *Special-Purpose Techniques for Additive Manufacturing*. Table collections with string names starting with **"**`ABQ_`**"** are reserved for special-purpose techniques and should not be used when programming your own user subroutines.

Postprocessing Simulation and In-Service Performance Validation

It is often necessary to simulate additional postprocessing operations performed on a printed part, such as wire electrical discharge machining (EDM) to remove a part from a build plate or to remove support structures, heat treatment, or other subsequent machining processes.

Cutting processes by wire EDM can be modeled using progressive removal of specified elements in the cutting region. Progressive element removal can be achieved by assigning a zero-valued material volume fraction to elements in a progressive element activation definition (see *Progressive Element Activation*). Alternatively, you can specify elements to remove from the model using the model change technology (see *Removing Elements*).

You can also apply heat treatment thermal cycles to simulate the reduction of residual stresses in the part due to annealing or other thermally induced phase transformations.

Furthermore, you can apply in-service loading to a printed part to account for residual stress effects from the manufacturing history. Abaqus/Standard offers various options to model part failure and part durability that can be incorporated into subsequent in-service performance validations.

References

- ISO / ASTM52900-15, "Standard Terminology for Additive Manufacturing-General Principles-Terminology," ASTM International, West Conshohocken, PA, , 2015.

Toolpath-Mesh Intersection Module

The toolpath-mesh intersection module computes the geometric intersections between various toolpaths used in additive manufacturing processes and a finite element mesh of the part to be manufactured.

In this section:

- *Toolpath-Mesh Intersection Toolpath Representations*
- *Point Toolpath-Mesh Intersection*
- *Infinite Line Toolpath-Mesh Intersection*
- *Box Toolpath-Mesh Intersection*
- *Scan Pattern–Mesh Intersection*
- *Activating and Using the Toolpath-Mesh Intersection Module*

Toolpath-Mesh Intersection Toolpath Representations

Products: Abaqus/Standard

References:

- *Additive Manufacturing Process Simulation*
- *Toolpath-Mesh Intersection Utility Routines*

Overview

A toolpath represents the motion of a given component of a machine tool such as a laser source, a recoater roller, or a wire-feed nozzle. A toolpath is defined by a geometric shape attached to a reference point that moves along a path. The path is defined by connecting a collection of points in space and time. An event series defines the collection of points. The first field defined in the event series describes a state of the tool, such as the laser power, the "on/off" state for a recoater roller, or a wire-feed nozzle. This field is assumed to be constant between two consecutive points. A zero-valued field indicates the "off" state of the tool.

Three shapes are considered for toolpath-mesh intersection: a point, an infinite line, and a box (see *Figure 1*). These shapes provide different levels of abstraction to characterize the shape of the tool depending on the particular application. In addition to these three shapes, a scan pattern that describes the idealized motion of a tool instead of the actual path of the motion can be used. The topics that follow list some of the quantities computed by the toolpath-mesh intersection module for each shape and a scan pattern. For a complete list of the quantities computed by the module, see the tables in *Data Retrieval Utility Routines*.

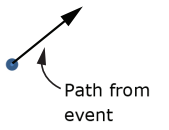
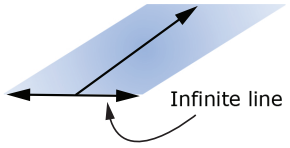
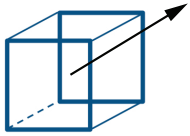
Shape	Point	Infinite Line	Box
Schematic			
Machine tool examples	Point lasers for coarse meshes	Recoater in powder-bed fabrication	Material deposition nozzle in polymer extrusion, wire-feed, laser blob for fine meshes, etc.

Figure 1: Point, infinite line, and box toolpaths.

Point Toolpath-Mesh Intersection

Products: Abaqus/Standard

References:

- *Toolpath-Mesh Intersection Toolpath Representations*

Overview

The point representation of the tool shape is useful for situations where the action zone of the tool is very small compared to the mesh size and can be idealized as a point; for example, when the laser beam radius is very small compared to the element size.

Figure 1 depicts intersections of a point toolpath with a finite element, E. The toolpath is defined by the path connecting points $(X_1, X_2, X_3, ..., X_n)$ at times $(t_1, t_2, t_3, ..., t_n)$. It is assumed that the tool travels at a constant velocity over a segment connecting two successive points in the path. The first field defined in the event series represents a state of the tool, such as the laser power. The field defined for a point X_n remains constant over the segment connecting X_n and X_{n+1}. All path segments where the tool is in the "on" state are required to be perpendicular to the global z_g-direction. For a given element, the toolpath-mesh intersection module computes the number of intersections of the toolpath, the coordinates of the start and end points (ξ_s and ξ_e, respectively, expressed in the element reference coordinate system), and the start and end times (t_s and t_e, respectively) for each intersection.

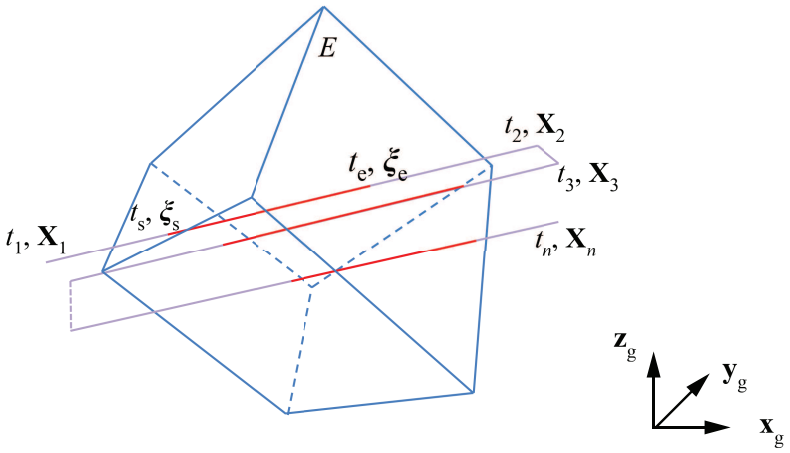


Figure 1: Point toolpath-mesh intersection.

Infinite Line Toolpath-Mesh Intersection

Products: Abaqus/Standard

References:

- *Toolpath-Mesh Intersection Toolpath Representations*

Overview

The infinite line representation of the tool shape is useful to describe the process of layer-by-layer material deposition, such as the action of the recoater roller in powder bed fabrication.

Figure 1 depicts intersections of an infinite line toolpath with a finite element, $\boldsymbol{E}$. The toolpath is defined by an infinite line attached to a reference point that is moving along the path connecting points $(\boldsymbol{X_1}, \boldsymbol{X_2}, \boldsymbol{X_3}, ..., \boldsymbol{X_n})$ such that the reference point is at $\boldsymbol{X_i}$ at time $\boldsymbol{t_i}$. It is assumed that the tool travels at a constant velocity over a segment connecting two successive points in the path, and the infinite line remains perpendicular to the segment. The first field defined in the event series represents a state of the tool, such as the "on/off" state of a recoater roller. The field defined for a point $\boldsymbol{X_n}$ remains constant over the segment connecting $\boldsymbol{X_n}$ and $\boldsymbol{X_{n+1}}$. All path segments when the tool is in the "on" state must be perpendicular to the global z_g-direction. For a given element, the toolpath-mesh intersection module computes the number of intersections, $\boldsymbol{m}$, of the toolpath and the volume fraction, $\boldsymbol{v_f}$, for each intersection. The volume fraction is equal to the ratio of the partial volume of the element below the z-plane defined by the motion of the infinite line following the path to the total volume of the element. The module also computes the area, A; the coordinate $\boldsymbol{X^a}$ with respect to the element reference coordinate system of the center of intersection of the z-plane and the element; as well as the area fractions, $\boldsymbol{a_f^i}$, below the z-plane for all sides (i = 1 to the number of side facets of the element) for each intersection.

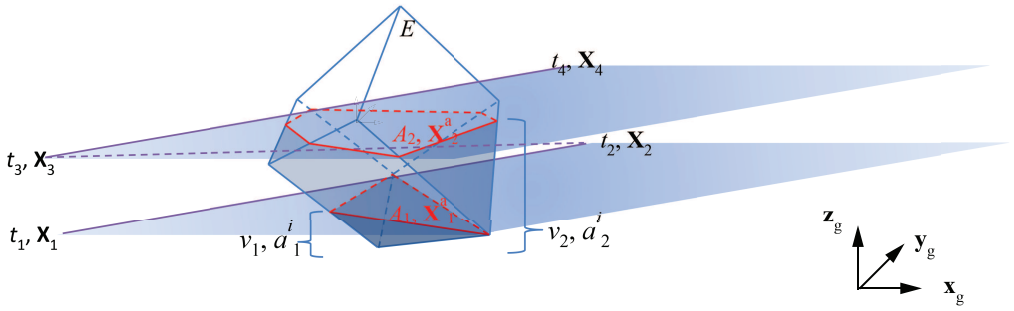


Figure 1: Infinite line toolpath-mesh intersection.

Box Toolpath-Mesh Intersection

Products: Abaqus/Standard

References:

- *Toolpath-Mesh Intersection Toolpath Representations*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

The box shape functionality is intended for situations where the action of the tool is best described as a spatially varying distribution.
Examples include modeling a Goldak's double ellipsoid heat source and polymer extrusion material deposition when using fine meshes (see *Special-Purpose Techniques for Additive Manufacturing*).

Intersections of a box toolpath and a mesh can be computed using two different algorithms or approaches; namely, the subsegment approach and the subelement approach. For both the subsegment and the subelement approaches, the box length in the local x_1-direction can be set to zero to obtain a rectangular-shaped toolpath.

Using the Subsegment Approach

Figure 1 depicts intersections of a box toolpath with a finite element, $\boldsymbol{E}$, using the subsegment approach. The toolpath is defined by a box attached to a reference point that is moving along the path connecting points $(\boldsymbol{X_1}, \boldsymbol{X_2}, \boldsymbol{X_3}, ..., \boldsymbol{X_n})$ such that the reference point is at $\boldsymbol{X_i}$ at time $\boldsymbol{t_i}$. A segment-specific local coordinate system is defined by the vectors $\mathbf{x_1}$, $\mathbf{y_1}$, and $\mathbf{z_1}$. Vector $\mathbf{x_1}$ is along the segment connecting two successive points, and $\mathbf{z_1}$ is a user-defined vector. The origin of this local system is at the start point of the segment. The box is oriented along the local coordinate system directions, and the center is at a constant user-defined offset, $\mathbf{F}$, from the reference point on the segment. The box lengths $\mathbf{L_1}$, $\mathbf{L_2}$, and $\mathbf{L_3}$ along the local directions are user defined. The box is divided into a user-defined number of smaller boxes. It is assumed that a subsegment starts at the center of each smaller box and is parallel to the main segment. A user-defined weight is associated with each subsegment. The weight is multiplied with the field associated with the main segment to obtain a field associated with the subsegment. The sum of the weights of all of the subsegments is usually equal to one. For a given element, the toolpath-mesh intersection module computes the number of intersections of subsegments with the element, the coordinates of the start and end points ($\boldsymbol{\xi}_s$ and $\boldsymbol{\xi}_e$, respectively, expressed in the element reference coordinate system), and the start and end times (t_s and t_e, respectively) for each intersection.

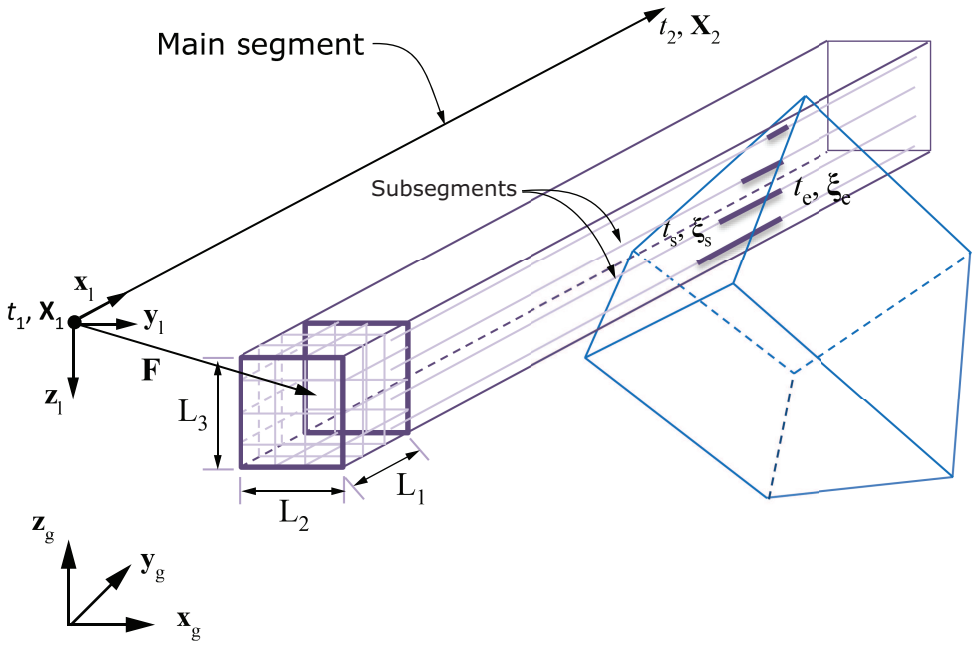


Figure 1: Box toolpath-mesh intersection using the subsegment approach.

Using the Subelement Approach

In the subelement approach, the box is not divided into smaller boxes. Instead, the element is subdivided into subelements of the same topology (see *Figure 2*). The number of subelement divisions of an element is set automatically by the module based on the element size and the minimum dimension of the box. The toolpath-mesh intersection module computes the number of subelements that have their centers inside the path of the box, the coordinates of the centers of the subelements, the volume of the subelements, and the start and end times of the box passing through the centers of the subelements.

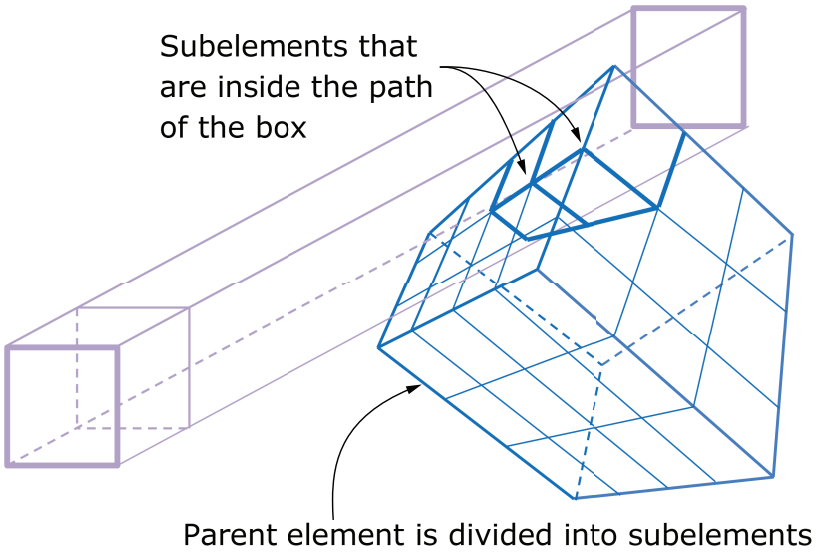


Figure 2: Box toolpath-mesh intersection using the subelement approach.

Scan Pattern–Mesh Intersection

Products: Abaqus/Standard

References:

- *Toolpath-Mesh Intersection Toolpath Representations*

Overview

A scan pattern is a representation of the movements of a tool that is moving or scanning regions of a cutting plane or slice.

Some additive manufacturing processes are characterized by a tool trajectory that follows a repetitive pattern in space; for example, powder bed fusion with the laser beam following a predefined island scanning strategy. In such cases, instead of describing individual trajectories of a toolpath, it is more effective to define a scan pattern that represents the idealized motion of a tool inside a part. The part being printed is divided into equally spaced (uniform thickness, $\boldsymbol{h}$) slices or cutting planes that are perpendicular to the build axis, $\boldsymbol{K}$ (see *Figure 1*(a) and *Figure 1*(b)). The build axis system *I*–*J*–*K* is a user-defined coordinate system that indicates the printing direction, *K*.

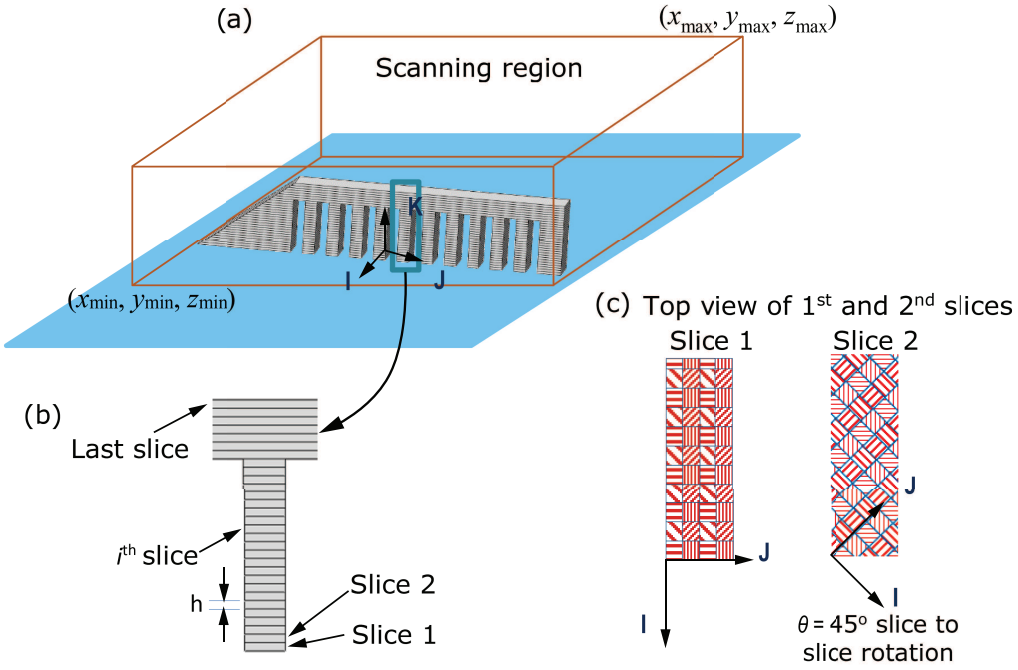


Figure 1: A scan pattern with four pattern patches.

The scan pattern consists of a rectangular unit cell (see *Figure 2*). The rectangular unit cell is repeated to cover the cutting plane. The rectangular unit cell consists of a number of smaller rectangular patches. Each patch can define a local angle, $\boldsymbol{\varphi}$, between the direction of the scanning motion of the tool and the *I*-axis. You can assign an eigenstrain tensor to each of the pattern patches representing the inelastic deformation induced by the process. You can define a scan pattern by defining extents of individual patches (x_{min}, y_{min}) and (x_{max}, y_{max}). All patches together must form a rectangular unit cell that must be situated entirely in the first quadrant of the *I*–*J* plane, and one corner of the cell must be at (0, 0).

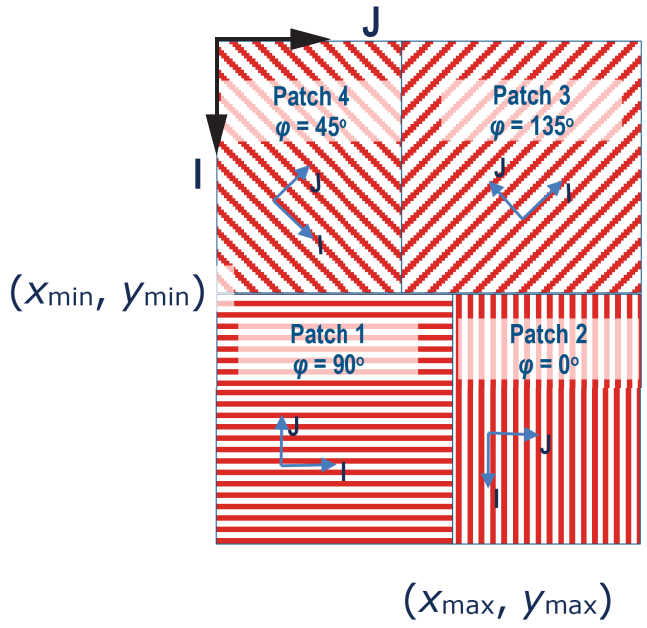


Figure 2: A scan pattern with four patches with local orientations rotated by 90°, 0°, 135°, and 45° with respect to the *I*-axis of the build axis system.

A scan pattern is active inside a scanning region. A scanning region is a build axis–oriented bounding box defined by its extent (x_{min}, y_{min}, z_{min}) and (x_{max}, y_{max}, z_{max}) (see *Figure 1*(a)). The height of a scanning region (z_{max}–z_{min}) must be an integral multiple of the thickness, h, of a slice. Multiple nonoverlapping scanning regions can be defined to cover the entire part. A different scan pattern can be active inside each scanning region. All scanning regions share the same build axis system. A layer-to-layer or slice-to-slice rotation angle, θ, can be defined. The scan pattern is rotated by $(i-1)\theta$ on the i^{th} slice for layer $i > 1$ (see *Figure 1*(c)).

For a given element, the toolpath-mesh intersection module computes the number of slices, m, inside the element in a given increment (see *Figure 3*). It finds which pattern patch contains the center of each slice in that element and the local orientation of that patch considering the layer-to-layer rotation, θ, and the local rotation, φ, of the scanning direction in that patch. The module also computes the partial volumes, v_f, of the element below each slice.

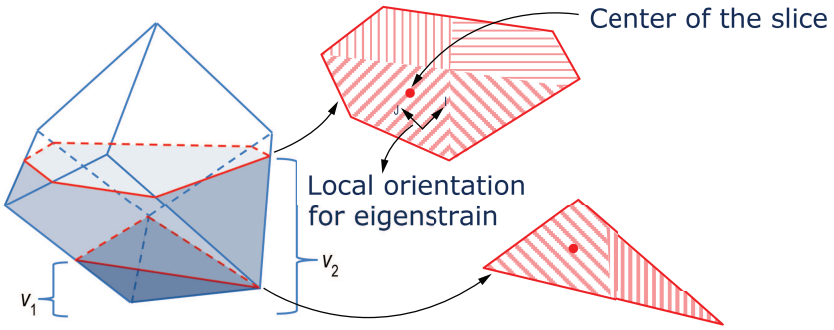


Figure 3: Scan pattern overlaid on an element.

Activating and Using the Toolpath-Mesh Intersection Module

Products: Abaqus/Standard

References:

- *Additive Manufacturing Process Simulation*
- *Toolpath-Mesh Intersection Utility Routines*
- *Special-Purpose Techniques for Additive Manufacturing*
- *UEPACTIVATIONSETUP*
- *UEPACTIVATIONVOL*
- *UEPACTIVATIONFACET*
- *UMDFLUX*

Overview

The toolpath-mesh intersection module enables simulations of a wide range of additive manufacturing processes. The functionality provides a high level of user control and customization using the *Toolpath-Mesh Intersection Utility Routines*, user subroutines, and table collections (see *About Table Collections, Parameter Tables, and Property Tables*).

This section describes a general workflow for the simulation of an additive manufacturing process using user subroutines and utility routines. The toolpath-mesh intersection module is also used in special-purpose techniques for common additive manufacturing processes (see *Special-Purpose Techniques for Additive Manufacturing*). You do not have to invoke the toolpath-mesh intersection module utilities or the user subroutines to use the special-purpose techniques.

Supported Elements

Linear and quadratic 4-, 5-, 6-, 8-, 10-, 15-, and 20-node solid elements and 3-, 4-, 6-, and 8-node shell elements with constant shell thickness are supported. For shell elements, the middle surface of the shell must be the reference surface.

Simulating Controlled Deposition of Raw Materials

You can use progressive element activation in a structural or a thermal analysis to simulate controlled deposition of raw materials. You can define a specific type of toolpath (for example, the infinite line or the box toolpath) that best approximates the sequence of the material deposition for the additive manufacturing process and compute the intersection of that toolpath with the finite element mesh. You can use the geometric information of the intersection to define the active/inactive status of elements at a given increment by invoking the toolpath-mesh intersection module utilities from the user subroutines that are associated with the progressive element activation.

Table 1: User subroutines associated with the progressive element activation.

UEPActivationSetup	Set up data needed for the toolpath-mesh intersection module for material deposition and the evolving free surface.
UEPActivationVol	Find the intersection information from the toolpath-mesh intersection module and define active/inactive status of elements and/or volume fraction of the element based on the intersection information. Specify the volume fraction increase for element activation. Optionally, specify material orientation and eigenstrain components to be applied upon activation.

UEPActivationFacet	Specify the facet area fraction of an element to apply a film or radiation condition during progressive element activation.

Simulating Laser-Induced Heating

You can define a moving heat flux to simulate laser-induced heating in a thermal analysis. You can define a specific type of toolpath that best approximates the motion and the nature of the heat source for the additive manufacturing process. You can use the geometric information of the intersection to define a heat flux in an element at a given increment by invoking the toolpath-mesh intersection module utilities from the user subroutines that are associated with the moving heat flux.

Table 2: User subroutines associated with the moving heat flux.

UMDFluxSetup	Set up data needed for the toolpath-mesh intersection module for the moving heat source.
UMDFlux	Prescribe the heat flux from the moving heat source.

Additive Manufacturing Process Simulation Workflow

The flowchart in *Figure 1* depicts a workflow of a typical additive manufacturing process simulation.

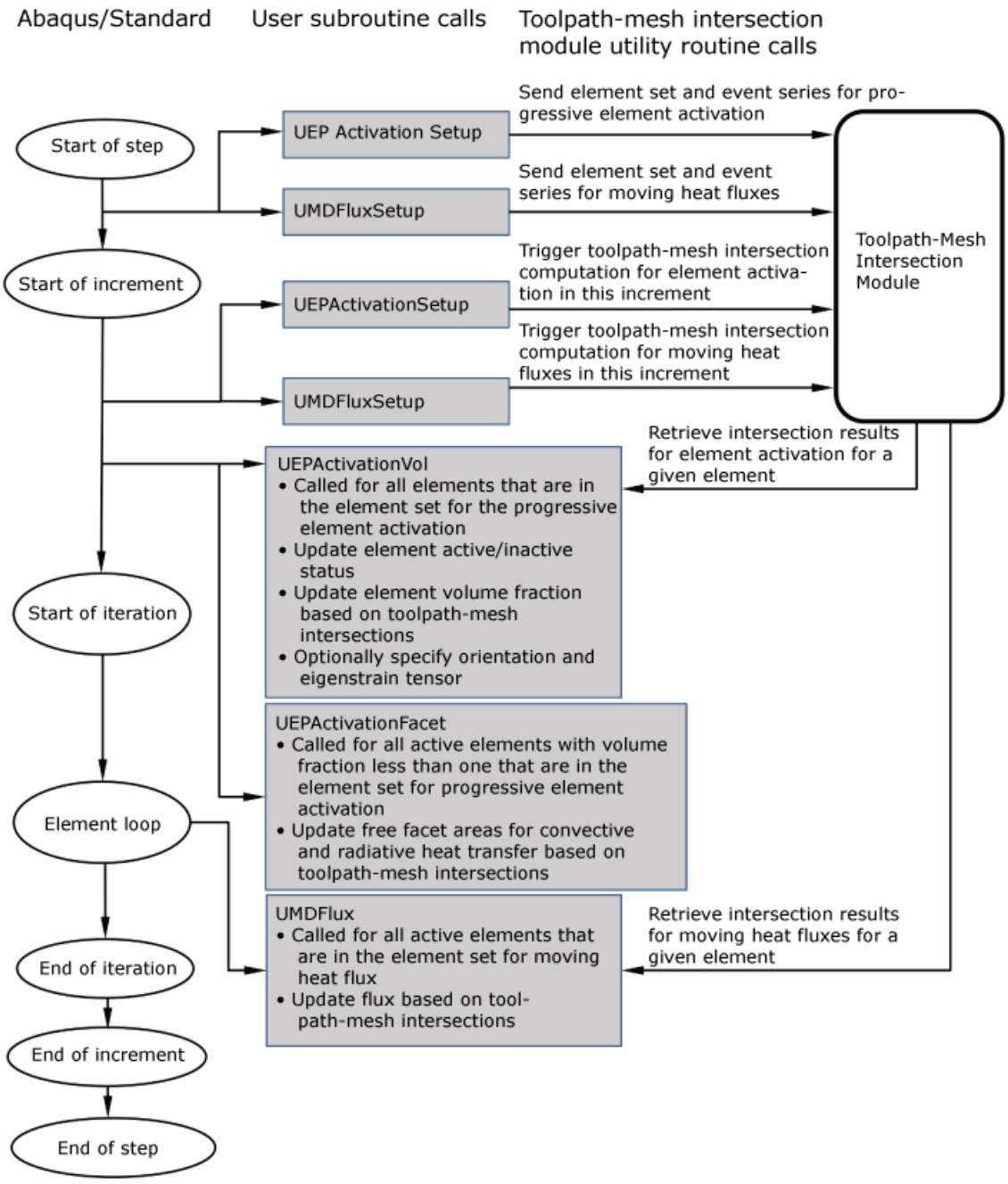


Figure 1: The workflow of a typical AM process simulation.

Thermomechanical Simulation of Additive Manufacturing Processes

Products: Abaqus/Standard

References:

- *Sequentially Coupled Thermal-Stress Analysis*
- *Progressive Element Activation*
- *Toolpath-Mesh Intersection Module*
- *Special-Purpose Techniques for Additive Manufacturing*
- *Thermal Loads*
- *About Table Collections, Parameter Tables, and Property Tables*
- *About Event Series*
- *Sequential thermomechanical analysis of a laser powder bed fusion build*
- *Sequential thermomechanical analysis of a directed energy deposition build*
- *Heat energy balance*

Overview

Abaqus/Standard offers a general framework for thermomechanical simulations that allows you to define appropriate boundary conditions, loads, interactions, constraints, and temperature-dependent material properties.
This section provides an overview of special techniques that are available for, but not limited to, thermomechanical simulations of additive manufacturing processes.

A thermal-stress analysis of additive manufacturing processes:

- allows exact specification in time and space of processing conditions, such as material deposition sequence, laser scan path, laser properties, and build environment;
- allows precise control over the fidelity of solutions for different simulation needs;
- can be followed by analyses of heat treatment applications, support removal, and/or mechanical performance tests, etc.; and
- is usually more accurate but more computationally expensive than an eigenstrain analysis.

Thermomechanical Simulation of Additive Manufacturing Processes

A sequential thermal-stress analysis of an additive manufacturing process consists of a transient heat transfer analysis of thermal loads introduced by the process on a printing part followed by a static structural analysis that is driven by the temperature field from the thermal analysis. The analysis can also include support structures (if required for the build) and a substrate where the part and support are built to consider their influences on thermal conduction, part distortions, and residual stresses.

For the heat transfer analysis, the following aspects of the process must be included in the simulation:

- Progressive material deposition: every additive manufacturing process adds new material with time.
- Progressive heating of the deposited material: for some additive manufacturing processes, the newly deposited layer of material gets heated up to the melting point, causing the material to fuse to an underlying layer or a substrate.
- Progressive cooling of the printing part: as a part is printed, its cooling surface is continuously evolving with time.

For the stress analysis,

- Temperatures from the heat transfer analysis are used to drive the stress analysis.
- Similar techniques of progressive material deposition as the heat transfer analysis can be used.
- Temperature-dependent material properties can be used for accurate stress results.

Progressive Element Activation in Thermomechanical Simulation

Progressive material deposition in additive manufacturing processes can be modeled using progressive element activation in Abaqus/Standard (see *Progressive Element Activation*). The parts and support structures can be activated progressively. During the analysis, any element that is defined as progressively activated is filled with material (completely or partially) or remains empty (inactive). You can control the activation and the volume fraction of the material for each element in each increment during an analysis with user subroutine *UEPACTIVATIONVOL*.

The material deposition sequence is defined in an event series (for example, in the form of time and spatial coordinates of a recoater sequence of events; see *About Event Series* for more details) and processed using user subroutine *UEPACTIVATIONVOL* to define how much material is added to an element in a given time increment. To simplify this process, you can access toolpath-mesh intersection utilities from within *UEPACTIVATIONVOL* to compute the change of the volume fraction of the material in each element and time increment taking into account the intersection of the finite element mesh with the specified event series data for material deposition (see *Toolpath-Mesh Intersection Module*). Optionally, you can update the material orientation to align it with the trajectory.

In addition, Abaqus/Standard provides a number of streamlined solutions for some of the most common additive manufacturing processes that do not require you to write user subroutines. These techniques are described in *Special-Purpose Techniques for Additive Manufacturing*.

Initial Temperature

During the additive manufacturing process of a part, newly deposited material comes in at a given initial temperature. You can define this initial temperature value by assigning initial conditions to the nodes of the mesh of the part to be printed (see *Defining Initial Temperatures*). Abaqus/Standard applies the initial temperature to the material inside these elements where they are first activated.

A stress analysis of a part-level thermomechanical simulation that uses a relatively coarse time increment and mesh is an exception to the previous statement. In this case you should assign an initial temperature representing a relaxation temperature above which thermal straining induces negligible thermal stress. This initial temperature is usually different from the initial temperature of the new material deposited in the manufacturing processes. For more information, see *Controlling the Scale of the Simulation and the Solution Fidelity*.

Displacement Output in Stress Analyses

When using progressive element activation in Abaqus/Standard, you can control the behavior of inactive elements to follow or not follow the deformation of active elements in the model (see *Controlling the Behavior of Inactive Elements*). The two behaviors are expected to produce similar results in the limit of small deformation, with the exception of displacements and rotations (U, UT, and UR).

An inactive element that follows the deformation, also referred to as a "quiet" element, is always present in the model and participates in the solution, but it produces a negligible contribution to the overall response. In this case a node attached to inactive elements can experience nonzero displacements before any of its attached elements are activated. The nodal output variables U, UT, and UR represent displacements and rotations measured from the beginning of an analysis, containing contributions of displacements during both inactive and active periods of a node. Abaqus/Standard also provides nodal output variables UACT, UTACT, and URACT corresponding to the displacements and rotations measured from the time when an element attached to the node is first activated.

An inactive element not following the deformation does not contribute to the stiffness of the model and does not participate in the solution. Any nodes attached to inactive elements remain in their initial position. In this case the nodal output variables UACT, UTACT, and URACT are the same as the output variables U, UT, and UR, respectively.

Regardless of the behavior chosen for inactive elements, the configuration of an element upon activation is usually different from the original configuration because nodes shared by active and inactive elements undergo displacements (see *Initial Configuration*). When an element is activated, the configuration at the time of activation is the reference for subsequent element calculations. Therefore, the output variable E represents strains measured from the time an element is activated.

Time Incrementation

The time increment used in analyses that include progressive element activation can influence the final results. Assume that two analyses are performed to simulate the material extrusion process of a bead of material that is modeled as a row of elements, with two different time increments: a small time increment activating one element per increment and a large time increment activating two elements per increment. The initial configuration of every second element is different in the two stress analyses, leading to different results of residual stresses and distortions. You can choose an appropriate time increment in thermomechanical analyses by performing a time-stepping convergence study.

Progressive Cooling via Convection and Radiation

With the deposition of new material during the additive manufacturing process, previously exposed material surfaces are covered, and new free surfaces are created with time. You can define surface convection and radiation on continuously evolving free surfaces (see *Specifying Element-Based Film Conditions on Evolving Faces of an Element in Abaqus/Standard* and *Specifying Element-Based Radiation Conditions on Evolving Faces of an Element in Abaqus/Standard*). Abaqus/Standard continually tracks the evolving free surfaces that reflect the current shape of a printing part at any given time in the build and applies film and radiation loadings only to those surfaces.

Progressive Heating by a Moving Heat Source

Many types of additive manufacturing processes involve the application of a moving heat source (laser, electron beam, welding torch, etc.) to fuse the raw material. You can define one or multiple nonuniform distributed heat sources in user subroutine *UMDFLUX* (see *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*).

The scanning path of a moving heat source is defined in an event series (for example, in the form of time, spatial coordinates and power of the heat source; see *About Event Series* for more details) and processed using user subroutine *UMDFLUX* to define each heating event that occurs on a given element in a given time increment. To simplify this process, you can access toolpath-mesh intersection utilities from within *UMDFLUX* to compute heat fluxes received by each element in each increment (see *Toolpath-Mesh Intersection Module*).

In addition, Abaqus/Standard provides a number of streamlined solutions to specify heat events that correspond to some of the most common heat sources in additive manufacturing processes that do not require you to write user subroutines. These techniques are described in *Special-Purpose Techniques for Additive Manufacturing*.

Controlling the Scale of the Simulation and the Solution Fidelity

Additive manufacturing process simulation is a multiscale problem in both time and space. You can control the scale of the simulation and the solution fidelity by choosing an appropriate time incrementation and mesh size. In general, you can consider two types of simulation at different ends of the fidelity spectrum: process-level simulations (high fidelity) versus part-level simulations (lower fidelity).

A detailed process-level simulation is performed by using a small time increment and a fine mesh with at least one element per printing layer thickness and a few elements across an action zone where active melting or fusion occurs. The simulation captures the rapidly evolving temperature and high-temperature gradients typically found within and near action zones and, thus, provides accurate predictions for both residual stresses and distortions. You can specify temperature-dependent thermal and mechanical material properties. You can model thermal energy release and absorption during melting and solidification in heat transfer analyses using latent heat definitions (see *Latent Heat*). For processes that use metallic materials, you can also model annealing effects or melting at a certain temperature in stress analyses (see *Annealing or Melting*). In addition, precise simulations of temperature histories facilitate the modeling of material phase transformations and microstructure evolutions during printing, which can be achieved with user subroutines (such as *UMAT*, *UMATHT* or *USDFLD*). For example, you can specify material properties for different phases of materials (such as powder, liquid, or different solid-phases) as functions of field variables or solution-dependent variables representing the material phase and evaluated inside *USDFLD* as a function of the temperature history (see *Specifying Field Variable Dependence* and *Specifying Material Data as Functions of Solution-Dependent Variables*). A process-level simulation can model detailed physics of additive manufacturing processes and provide accurate results, but it typically has a high computational cost and can be affected by convergence difficulties due to the use of temperature-dependent nonlinear material properties under rapidly changing temperature conditions.

A part-level simulation is performed by appropriately averaging (lumping) the time sequence of events and using coarser meshes. For example, the model could have an element size that is a few times larger than the physical printing layer thickness and use only one or several time increments for printing one element layer. The heat transfer analysis can usually capture far-field temperature evolutions (away from action zones) as long as the thermal energy balance of progressive heating and cooling is modeled correctly. However, the simulation may not capture local rapid temperature evolutions properly because the specified sequence of concentrated, fast-moving heat sources is lumped over both time and space. In other words, the temperature results do not usually contain an accurate history of melting and solidification. In this case, to model the annealing or melting effects in the stress analysis correctly, you must assign an initial temperature representing a relaxation temperature above which thermal straining induces negligible thermal stress in the printing part. Upon element activation, the relaxation temperature is the temperature from which the initial thermal contraction occurs. This relaxation temperature can be calibrated from experiments or detailed process-level simulations. The part-level simulation is computationally efficient for the prediction of distortions and stresses in printing parts with reasonable accuracy.

Resolving Convergence Difficulties

Thermomechanical simulations of additive manufacturing processes are usually nonlinear due to temperature-dependent material behavior, progressive element activation, or other factors. Common convergence issues and recommendations on how to resolve them are discussed below.

In the heat transfer analysis, convergence difficulties are usually caused by the nonlinearity of material properties.

- If temperature-dependent conductivity is specified for the material, the system of equations is unsymmetric. In this case using the unsymmetric matrix storage and solution scheme (see *Matrix Storage and Solution Scheme in Abaqus/Standard*) may help to improve convergence.
- When latent heat effects are considered, they can introduce severe nonlinearity in the analysis and have a negative impact on convergence. Abaqus/Standard provides options to reduce the nonlinearity by specifying a smooth transition or extending the solidus-liquidus temperature interval (see *Defining a Smooth Latent Heat Transition*).

In the stress analysis, convergence difficulties can be encountered during the activation of elements.

- Elements may distort excessively before they are activated and cause convergence difficulties. In such situations you should specify that inactive elements follow the deformation to prevent excessive element distortion (see *Controlling the Behavior of Inactive Elements*).
- The initial thermal strain experienced by an element upon activation (due to the difference between the current temperature and the specified initial temperature) can lead to convergence difficulties that cannot be resolved

by reducing the time increment. To overcome this issue, Abaqus/Standard provides an option to ramp the thermal strains over a period of time instead of applying them instantaneously upon activation (see *Applying Initial Thermal Strains*). Ramping thermal strains can influence the accuracy of the analysis results. For example, if the thermal strains of elements in a layer are not fully ramped when the next layer of elements is activated, the strain-free configuration of the newly activated elements is different from the case when thermal strains are fully ramped. You should use a ramping time constant smaller than the required time for processing one layer.

- If the material definition includes plasticity, the analysis may iterate excessively due to the extrapolation scheme used to speed up the solution. You can prevent this issue by turning off extrapolation (see *Incrementation in Abaqus/Standard*).

Input File Template

The following template shows the input for the heat transfer analysis `heat.inp`:

```
*HEADING
…
** Choose the uncoupled heat transfer element type
*ELEMENT, TYPE=DC3D8
** Define material deposition motion and/or heat source scan path
*EVENT SERIES TYPE
*EVENT SERIES
** Define additional information for material deposition and/or moving heat sources
*PARAMETER TABLE TYPE
*PROPERTY TABLE TYPE
*TABLE COLLECTION
*PARAMETER TABLE
*PROPERTY TABLE
** Define elements that can be activated during an analysis
*ELEMENT PROGRESSIVE ACTIVATION
…
*PHYSICAL CONSTANTS, ABSOLUTE ZERO=θ^Z, STEFAN BOLTZMANN=σ
*INITIAL CONDITIONS, TYPE=TEMPERATURE
…
*STEP, UNSYMM=YES
*HEAT TRANSFER
…
** Turn on progressive element activation in the step
*ACTIVATE ELEMENTS
table collection for material deposition (if defined)
** Apply thermal loads and boundary conditions
*BOUNDARY
Data line to define nonzero boundary conditions on temperature degree of freedom
*FILM
element number or element set name, FFS or FFSNU, θ^0, h
*RADIATE
element number or element set name, RFS, θ^0, ϵ
*DFLUX
element set name, MBFNU, blank entry, table collection for a heat source (if defined)
…
** Write all nodal temperatures to the output database file, heat.odb
*OUTPUT, FIELD
*NODE OUTPUT
 NT
*END STEP
```

The following template shows the input for the subsequent static structural analysis:

```
*HEADING
…
** Choose the continuum element type compatible with the heat transfer element type used
*ELEMENT, TYPE=C3D8
…
** Define material deposition motion
*EVENT SERIES TYPE
*EVENT SERIES
** Define additional information for material deposition
*PARAMETER TABLE TYPE
*PROPERTY TABLE TYPE
*TABLE COLLECTION
*PARAMETER TABLE
*PROPERTY TABLE
** Define elements that can be activated during an analysis
*ELEMENT PROGRESSIVE ACTIVATION
…
*INITIAL CONDITIONS, TYPE=TEMPERATURE
*BOUNDARY
Data lines to specify zero-valued boundary conditions on displacement degree of freedom
…
*STEP, EXTRAPOLATION=NO
*STATIC
…
** Read in all nodal temperatures from the output database file, heat.odb
*TEMPERATURE, FILE=heat.odb
** Turn on progressive element activation in the step
*ACTIVATE ELEMENTS
table collection for material deposition (if defined)
** Apply structural loads and boundary conditions
…
*END STEP
```

Eigenstrain-Based Simulation of Additive Manufacturing Processes

Products: Abaqus/Standard

References:

- *Static Stress Analysis*
- *Progressive Element Activation*
- *Toolpath-Mesh Intersection Module*
- *Special-Purpose Techniques for Additive Manufacturing*
- *About Table Collections, Parameter Tables, and Property Tables*
- *About Event Series*
- *Pattern-based eigenstrain analysis*

Overview

Abaqus/Standard offers a general framework for eigenstrain-based simulations of additive manufacturing processes. This section provides an overview of special techniques that are available for, but not limited to, eigenstrain-based simulations of additive manufacturing processes. These techniques can be applied to other processes, such as welding.

An eigenstrain analysis of additive manufacturing processes:

- is a computationally efficient method for the prediction of part-level distortion and residual stresses introduced during the additive manufacturing process;
- consists of a single stress analysis with a predefined set of eigenstrains that are applied to each element upon activation and that represent the inelastic deformation induced by the processes;
- simplifies the definition of the problem by eliminating the need to specify detailed processing conditions;
- is generally a more approximate solution with a shorter modeling and simulation time than a thermal-stress analysis; and
- can be followed by analyses of support removal, and/or mechanical performance tests, etc.

About Eigenstrain

Residual stresses in mechanical parts are stresses that exist in the absence of externally applied loads. Almost all manufacturing processes, including additive manufacturing, introduce residual stresses into mechanical parts. Residual stresses are sometimes introduced intentionally to improve the in-service response, such as in prestressed concrete slabs used in bridge construction. However, manufacturers often try to minimize residual stresses because they can cause fracture during the manufacturing process, lead to unwanted distortions, and significantly impact fatigue behavior. Three primary classes of manufacturing effects lead to residual stresses:

- Mechanical (for example, inelastic deformation);
- Thermal (for example, nonuniform thermal expansion or incompatible thermal strains generated during melting and solidification in the process zone); and
- Changes in material microstructure (for example, phase transformations).

Eigenstrain (also referred to as inherent strain, assumed strain, or "stress-free" strain) is an engineering concept used to account for all sources of inelastic deformation that lead to residual stresses and distortions in manufactured components. Thermal strains are a subset of eigenstrains.

In a linear elastic deformation, the stress induced by an eigenstrain can be represented as

$$\boldsymbol{\sigma} = \mathbf{D}^{el} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}^{*}) = \mathbf{D}^{el} : \boldsymbol{\varepsilon}^{el},$$

where

$\boldsymbol{\sigma}$ is the Cauchy stress;

$\mathbf{D}^{el}$ is the elastic matrix;

$\boldsymbol{\varepsilon}$ is the total strain;

$\boldsymbol{\varepsilon}^{*}$ is the eigenstrain; and

$\boldsymbol{\varepsilon}^{el}$ is the elastic strain.

Using constitutive equations (such as the one shown above) eigenstrains can be used to compute residual stresses coming from mechanical, thermal, and microstructural sources.

An eigenstrain in three dimensions is represented as a standard strain tensor with six components:

$$\boldsymbol{\varepsilon}^{*} = \begin{Bmatrix} \varepsilon^{*}_{11} \\ \varepsilon^{*}_{22} \\ \varepsilon^{*}_{33} \\ \varepsilon^{*}_{12} \\ \varepsilon^{*}_{13} \\ \varepsilon^{*}_{23} \end{Bmatrix}.$$

The components of the eigenstrain tensor are functions of many factors, including material properties, manufacturing processes, and thermal history. Various methods can be used to determine appropriate eigenstrains for a given process:

- Destructive and nondestructive tests of manufactured parts.
- Numerical simulation.
- Analytical formulas for simple scenarios.

Once an appropriate eigenstrain field has been determined, it can be applied in an eigenstrain analysis to predict the distortions and residual stresses in an additive manufactured part.

Eigenstrain-Based Simulation of Additive Manufacturing Processes

An eigenstrain analysis of an additive manufacturing process consists of a single static stress analysis of a printing part with a predefined field of eigenstrains that are applied to each element upon activation representing the inelastic deformation induced by the process. These inelastic deformations become the main source of residual stresses and overall part distortion; therefore, the objective of an eigenstrain analysis is to predict distortions and residual stresses in the part. Eigenstrains applied to a newly deposited layer can induce residual stresses and distortion on layers underneath. *Figure 1* shows a simple example of an additive manufacturing process of a two-layer build with the following conditions:

1. The first layer is added.
2. The first layer is unconstrained—it contracts when negative eigenstrains are applied.
3. The second layer is added on top of the first layer and bonded to the first layer.
4. The contraction of the second layer is constrained by the bonding of the first layer, causing the part to distort and inducing residual stresses.

The eigenstrain analysis can also include support structures (if required for the build) and a substrate where the part and support are built to consider their influences on part distortions and residual stresses. In general, an

eigenstrain analysis provides a more approximate solution than a thermal-stress analysis. However, because only a static procedure is required, an eigenstrain analysis often has a shorter run time.

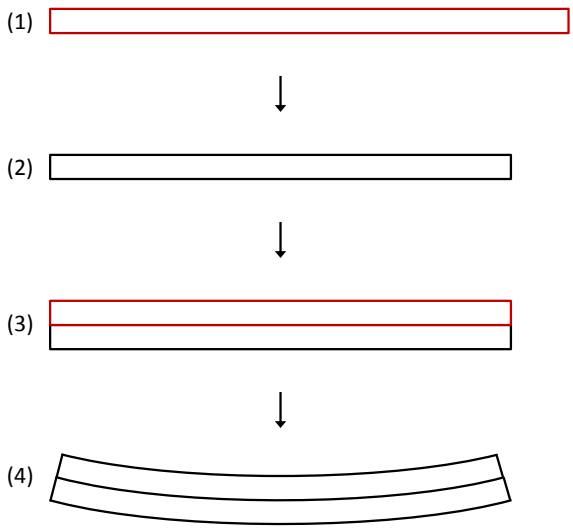


Figure 1: Distortion due to eigenstrains in a two-layer additive manufacturing process.

Progressive Element Activation and Eigenstrains Application

Material deposition in the additive manufacturing processes is modeled in Abaqus/Standard by progressive element activation (see *Progressive Element Activation*). Elements are activated in either a full or partially full state. In each increment during an analysis, you can use user subroutine *UEPACTIVATIONVOL* to control the element activation and the volume fraction of material added to each element and to define the eigenstrain tensor associated with the new material (see *Applying Eigenstrains*). Abaqus/Standard automatically applies the eigenstrain to the element, introducing residual stress in the element.

By using user subroutine *UEPACTIVATIONVOL*, you have complete control over the element activation sequence and the eigenstrain values to apply. You can access toolpath-mesh intersection utilities that are specially designed to define and apply eigenstrains for common additive manufacturing processes (see *Toolpath-Mesh Intersection Module*). Two types of eigenstrain analyses support this functionality: trajectory-based and pattern-based.

Abaqus/Standard also provides streamlined solutions for common trajectory-based and pattern-based eigenstrain analyses that do not require you to write user subroutines. These techniques are described in *Special-Purpose Techniques for Additive Manufacturing*.

Trajectory-Based Eigenstrain Analysis

A trajectory-based eigenstrain analysis activates elements and applies eigenstrains based upon a specified trajectory of new material being fused or bonded to the underlying layer. For example, the trajectory of a powder bed fusion process is the same as the heat source scan path, and the trajectory for directed energy deposition and material extrusion processes is the nozzle path. The trajectory is defined using an event series (in the form of time, spatial coordinates, and user-defined data of events; see *About Event Series* for more details) and processed directly by the toolpath-mesh intersection module. In user subroutine *UEPACTIVATIONVOL*, you can call the relevant toolpath-mesh intersection utilities to obtain information about the change of volume fraction of material and the eigenstrain values to assign to each element in each increment. Optionally, you can update the material orientation to align it with the trajectory. The analysis is similar to the stress analysis in thermomechanical simulations, except that it is driven by eigenstrain loadings instead of temperature results for the thermal analysis.

Pattern-Based Eigenstrain Analysis

A pattern-based eigenstrain analysis activates elements layer by layer and applies eigenstrain based on a specified in-plane eigenstrain pattern for each layer. An eigenstrain pattern is a domain that is partitioned by a "quilt" of one or more patches. Each patch is an area that contains a specific value of eigenstrains or a rotation angle of eigenstrains as a result of a specific trajectory in that area. For example, the eigenstrain patterns for powder bed fusion processes are related to the in-plane scan pattern of the heat source, and the eigenstrain patterns for directed energy deposition processes and material extrusion processes are related to the in-plane moving pattern of the nozzle. A pattern-based eigenstrain analysis does not require you to define a trajectory. The analysis considers layer-by-layer building sequences and ignores the detailed sequences of material deposition or scanning within layers. You define parameters and properties of eigenstrain patterns using table collections (see *About Table Collections, Parameter Tables, and Property Tables*) and access them using user subroutine *UEPACTIVATIONVOL*. In the user subroutine you can activate elements in a layer-by-layer fashion, call the toolpath-mesh intersection utilities to identify which eigenstrain patch an element in the last activated layer belongs to, and apply the eigenstrains to the element. You can also update the material orientation, such as aligning it with the rotation angle of the eigenstrain of the patch.

Partial Element Activation in Eigenstrain-Based Simulation

Eigenstrain-based simulations in Abaqus/Standard are supported with full and partial element activation. In the case of partial element activation the volume fraction of material added can be arbitrary; however, in practice the value should be larger than a small threshold value to avoid numerical singularity problems. Full activation is a special case of partial activation when the volume fraction of material added is restricted to 1.0.

For partial activation, when new material is added in an increment, both the old and new material contribute to the stress response of the material. In general, the two materials might be in different states; therefore, the homogenized values of state variables are used to compute stresses. Abaqus/Standard homogenizes the variables using the rule of mixtures in which the variables are computed using the volume weighed average values. For example, for the linear elastic material model the response is computed from:

$$\boldsymbol{\sigma} = V^f \mathbf{D}^{el} : \boldsymbol{\varepsilon}^{el},$$

where

V^f is the volume fraction of the material in the element; and

$\boldsymbol{\varepsilon}^{el}$ is the homogenized elastic strain.

In general, the new material is added with the eigenstrain prescribed. In this case the homogenized elastic strain is computed from the relation

$$\boldsymbol{\varepsilon}^{el} = \Delta\boldsymbol{\varepsilon} + \frac{V_t^f}{V^f}\boldsymbol{\varepsilon}_t^{el} - \frac{\Delta V^f}{V^f}\Delta\boldsymbol{\varepsilon}^*,$$

where

V_t^f is the volume fraction of the material in the element in the previous increment;

ΔV^f is the volume fraction of the material added to the element ($\Delta V^f = V^f - V_t^f$);

$\Delta\boldsymbol{\varepsilon}$ is the total strain increment;

$\Delta\boldsymbol{\varepsilon}^*$ is the eigenstrain in the material added; and

$\boldsymbol{\varepsilon}_t^{el}$ is the elastic strain at the end of the previous increment.

The configuration at which the new material is added is stress free only if no eigenstrain is prescribed. If the eigenstrain is specified, it causes a sudden increase of stress that does not decrease when the time increment is cut. This behavior can cause convergence difficulties, particularly when geometric nonlinearities are taken into account and nonlinear material models, such as the elastic-plastic model, are used. In such cases Abaqus/Standard provides the option of ramping up the eigenstrain linearly in the specified time interval (see *Progressive Element Activation*). However, you must use caution when choosing the ramping time value; it should be small relative to the analysis so that the results are not strongly affected.

Displacement Output

When using progressive element activation in Abaqus/Standard, you can control the behavior of inactive elements to follow or not follow the deformation of active elements in the model (see *Controlling the Behavior of Inactive Elements*). The two behaviors are expected to produce similar results in the limit of small deformation, with the exception of displacements and rotations (U, UT, and UR).

An inactive element that follows the deformation, also referred to as a "quiet" element, is always present in the model and participates in the solution, but it produces a negligible contribution to the overall response. In this case a node attached to inactive elements can experience nonzero displacements before any of its attached elements are activated. The nodal output variables U, UT, and UR represent displacements and rotations measured from the beginning of an analysis, containing contributions of displacements during both inactive and active periods of a node. Abaqus/Standard also provides nodal output variables UACT, UTACT, and URACT corresponding to the displacements and rotations measured from the time when an element attached to the node is first activated.

An inactive element not following the deformation does not contribute to the stiffness of the model and does not participate in the solution. Any nodes attached to inactive elements remain in their initial position. In this case the nodal output variables UACT, UTACT, and URACT are the same as the output variables U, UT, and UR, respectively.

Regardless of the behavior chosen for inactive elements, the configuration of an element upon activation is usually different from the original configuration because nodes shared by active and inactive elements undergo displacements (see *Initial Configuration*). When an element becomes active, the configuration at the time of activation is the reference for subsequent element calculations. Therefore, the output variable E represents strains measured from the time an element is activated.

Time Incrementation

The time increment used in eigenstrain analyses can influence the final results. Assume that two eigenstrain analyses are performed to activate a row of elements using different time increments: a small time increment activating one element per increment and a large time increment activating two elements per increment. The initial configuration of every second element is different between the two analyses, leading to different results of residual stresses and distortions. You can choose an appropriate time increment for a trajectory-based eigenstrain analysis by performing a time stepping convergence study. For a pattern-based eigenstrain analysis, it is recommended that you use a time increment less than the time taken to process one element layer.

Resolving Convergence Difficulties

In eigenstrain analyses convergence difficulties can occur when elements are activated and the eigenstrain is applied.

- Elements may distort excessively before they are activated and cause convergence difficulties. In such situations you should specify that inactive elements follow the deformation to prevent excessive element distortion (see *Controlling the Behavior of Inactive Elements*).
- The analysis can have convergence issues if large eigenstrains are applied instantaneously upon element activation. This issue cannot be resolved by reducing the time increment. To overcome this issue,

Abaqus/Standard provides an option to ramp the eigenstrains over a period of time instead of applying them instantaneously upon activation (see *Applying Eigenstrains*). Ramping eigenstrains can influence the accuracy of the analysis results. For example, if the eigenstrains of elements in a layer are not fully ramped when the next layer of elements is activated, the strain-free configuration of the newly activated elements is different from the case when the eigenstrains are fully ramped. You should use a ramping time constant smaller than the time increment required for processing one layer.

- If the material definition includes plasticity, the analysis may iterate excessively due to the extrapolation scheme used to speed up the solution. You can prevent this issue by turning off extrapolation (see *Incrementation in Abaqus/Standard*).

Input File Template

The following template shows the input for an eigenstrain analysis:

```
*HEADING
…
** Define material deposition motion (for trajectory-based eigenstrain analyses)
*EVENT SERIES TYPE
*EVENT SERIES
** Define additional information (for example, predefined eigenstrain libraries, etc.)
*PARAMETER TABLE TYPE
*PROPERTY TABLE TYPE
*TABLE COLLECTION
*PARAMETER TABLE
*PROPERTY TABLE
** Define elements that can be activated during an analysis
*ELEMENT PROGRESSIVE ACTIVATION
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions on displacement degree of freedom
…
*STEP, EXTRAPOLATION=NO
*STATIC
…
** Turn on progressive element activation in the step
*ACTIVATE ELEMENTS
table collection (if defined)
** Apply structural loads and boundary conditions
…
*END STEP
```

Special-Purpose Techniques for Additive Manufacturing

Special-purpose techniques for common additive manufacturing processes are described in this section.

The functionality in Abaqus/Standard for additive manufacturing processes simulation is developed on a user subroutine infrastructure and keyword interface that provides a high degree of control and customization. Abaqus/Standard provides very general capabilities for the simulation of additive manufacturing processes using either thermomechanical or eigenstrain analyses (see *Thermomechanical Simulation of Additive Manufacturing Processes* and *Eigenstrain-Based Simulation of Additive Manufacturing Processes*).

In addition, a number of special-purpose techniques are available for simulation of common AM processes that do not require you to write user subroutines. These techniques are implemented as "internal" user subroutines in Abaqus using the same user subroutine infrastructure and keyword interface. These special-purpose techniques are accessed by using table collections with string names starting with either `"ABQ_AM"` or `"ABQ_EIG"`. Table collections with string names starting with `"ABQ_AM"` and `"ABQ_EIG"` are reserved for special-purpose techniques and should not be used when programming your own user subroutines.

You can use these special-purpose techniques to define:

- Progressive element activation in a structural or a thermal analysis to simulate controlled deposition of raw materials. You can simulate a layer-upon-layer raw material deposition by a recoater or roller blade used in powder bed–type processes and a bead type deposition sequence for materials injected through nozzles used in directed energy deposition processes.
- Moving heat fluxes to model laser-induced heating in a thermal analysis.

To activate these special-purpose techniques, you must define the pertinent manufacturing process parameters in table collections that follow prescribed conventions. The sections that follow describe these conventions. The table collections must adhere to the naming conventions and include parameter and property tables of predetermined types. You can refer to these table collections from progressive element activation and/or distributed heat flux definitions. Abaqus activates elements and applies heat fluxes automatically using the specified process parameters.

In this section:

- *Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Trajectory-Based Method*
- *Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Pattern-Based Method*
- *Thermomechanical Analysis of FDM- and LDED-Type Additive Manufacturing Processes*
- *Eigenstrain-Based Analysis of Additive Manufacturing Processes*
- *Machining and Material Removal Process*
- *Parameter Table Type Reference*
- *Event Series and Property Table Type Reference*

Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Trajectory-Based Method

Products: Abaqus/Standard

References:

- *Progressive Element Activation*
- *Sequential thermomechanical analysis of a laser powder bed fusion build*
- *Heat energy balance*
- *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*
- **EVENT SERIES*
- **EVENT SERIES TYPE*
- **TABLE COLLECTION*
- **PARAMETER TABLE*
- **PARAMETER TABLE TYPE*
- **PROPERTY TABLE*
- **PROPERTY TABLE TYPE*

Overview

Special-purpose techniques and user subroutines are available to define the relevant process parameters for material deposition and heat sources.
These internal built-in user subroutines are accessed by starting names and types with `"ABQ_"` as described below.

In a powder bed–type additive manufacturing (AM) process, such as selective laser sintering (SLS) and stereolithography (SLA), a recoater or a roller blade deposits a single layer of raw material. Then, a high-powered laser scans a single cross-section of the part over the layer of raw material to fuse it with the previously laid layer underneath (see *Figure 1*). The layer-upon-layer raw material deposition is simulated by progressive element activation in a structural or a thermal analysis, and the laser-induced heating is simulated by a moving heat flux in a thermal analysis.

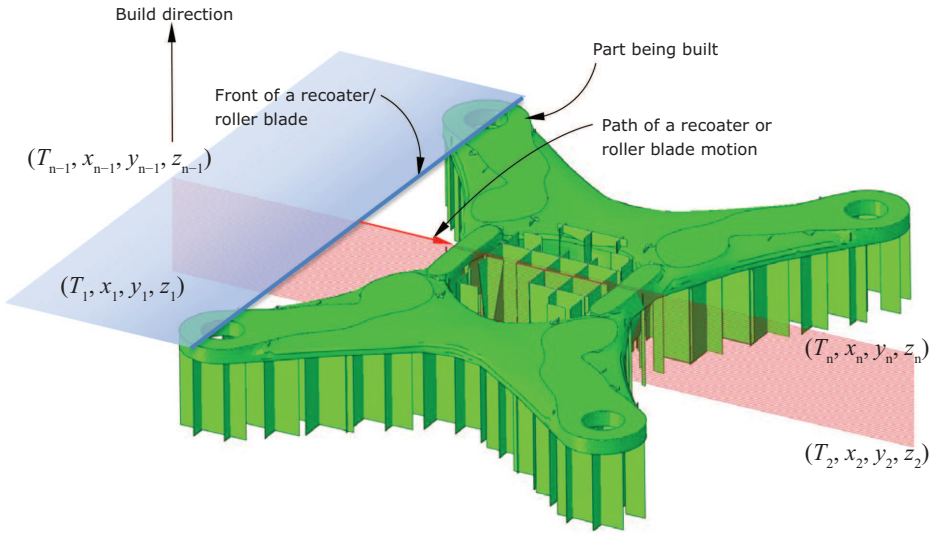


Figure 1: Recoater or roller blade motion in a powder bed–type AM process.

Specifying Progressive Element Activation

The layer-by-layer deposition of raw material from a recoater or roller blade is simulated using progressive element activation in a structural or a thermal analysis. The following steps are required to define the deposition process completely:

- Define the motion of the center point of the recoater in an event series following the convention for infinite line toolpath-mesh intersection (see *Infinite Line Toolpath-Mesh Intersection*).
- Create a table collection with a name that begins with `"ABQ_AM"`. The table collection must contain a parameter table of type *"ABQ_AM_MaterialDeposition"*.
- In the parameter table, include a reference to the event series for the material deposition, and set the deposition process type to `"Roller"`.
- Refer to the table collection in the progressive element activation.

Abaqus/Standard activates elements automatically according to the specified material deposition sequence.

A dedicated collection of parameter table, property table, and event series types is available to include all of the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all of the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage: Use the following options to define progressive element activation in a thermal or a structural analysis:

**EVENT SERIES*, NAME=*event series name*, TYPE=*ABQ_AM_MaterialDeposition*

$t_1, x_1, y_1, z_1, 1$

$t_2, x_2, y_2, z_2, 0$

$t_3, x_3, y_3, z_3, 1$

$t_4, x_4, y_4, z_4, 0$

...

$t_n, x_n, y_n, z_n, 0$

**TABLE COLLECTION*, NAME="ABQ_AM_*table collection name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition"*
event series name, "Roller"
**PARAMETER TABLE*, TYPE=*ABQ_AM_MaterialDeposition_Advanced*
"Full" or "Partial", *min volume fraction threshold for partial activation*, *max volume fraction threshold for partial activation*, *volume fraction threshold for full activation*
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME=*activation name*, ELSET=*element set name*
...
**STEP*
**STATIC* or **HEAT TRANSFER*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*
"ABQ_AM_*table collection name*"
...

Specifying a Concentrated Moving Heat Source

You can approximate the laser spot as a concentrated moving heat flux if the size of the finite elements used in a thermal analysis is significantly larger than the size of the laser spot (see *Figure 2*). The following steps are required to define the concentrated moving heat source completely:

- Define the scanning trajectory and power of the laser in an event series following the convention for point toolpath-mesh intersection (see *Point Toolpath-Mesh Intersection*).
- Create a table collection with a name that begins with `"ABQ_AM"`. The table collection must contain a parameter table of type *"ABQ_AM_MovingHeatSource"*.
- In the parameter table, include a reference to the event series for the point heat source and set the heat source type to `"Concentrated"`.
- Refer to the table collection in the distributed load definition.

Abaqus/Standard computes and applies moving heat fluxes to each element automatically according to the specified scanning trajectory.

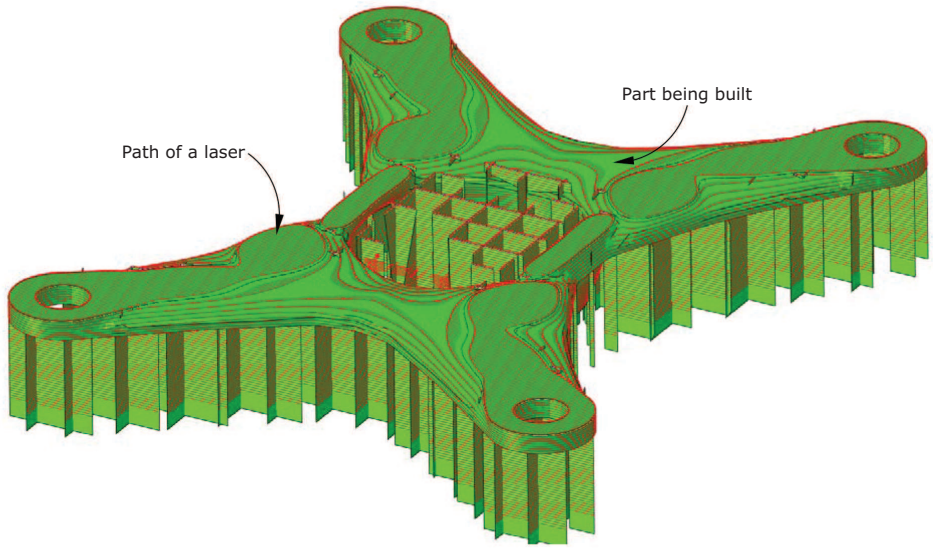


Figure 2: Path of a laser heat source.

Input File Usage: Use the following options to define a concentrated moving heat flux:

**EVENT SERIES*, NAME=*event series name*, TYPE=*"ABQ_AM_PowerMagnitude"*
data lines to define the motion of the laser spot
**TABLE COLLECTION*, NAME="ABQ_AM_*table collection name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MovingHeatSource"*
event series name, "Concentrated"
**PROPERTY TABLE*, TYPE=*"ABQ_AM_AbsorptionCoeff"*
absorption coefficient
...
**STEP*
**HEAT TRANSFER*
...
**DFLUX*
element set name, MBFNU, ,"ABQ_AM_*table collection name*"
...

Specifying a Moving Heat Source with a Goldak Distribution

If the size of the finite elements used in a thermal analysis is comparable to the size of the laser spot, the laser power, Q, can be distributed over a volume based on the Goldak (double ellipsoid) rule of laser energy distribution. *Figure 3* shows the Goldak expression for energy distribution, q, from a laser source. The local x-axis indicates the laser motion direction defined by an event series segment. The following steps are required to define the moving heat source completely:

- Define the trajectory of the laser spot in an event series similar to the definition of the concentrated moving heat source.
- Create a table collection with a name that begins with `"ABQ_AM"`. The table collection must contain a parameter table of type *"ABQ_AM_MovingHeatSource"* and a parameter table of type *"ABQ_AM_MovingHeatSource_Goldak"*.
- In the parameter table of type *"ABQ_AM_MovingHeatSource"* include a reference to the event series for the moving heat source and set the heat source type to `"Goldak"`.
- In the parameter table of type *"ABQ_AM_MovingHeatSource_Goldak"* define the parameters of the Goldak distribution (`a`, `b`, `cf`, `cr`, `ff`, and `fr`). For laser applications, the geometric parameters in the Goldak expression (that is `a`, `b`, `cf`, and `cr`) are typically calibrated to match the melt pool geometry observed experimentally. The parameters `ff` and `fr` represent the front and rear distribution fractions, respectively, and are used to account for the asymmetry in the heat distribution. Energy conservation requires that `ff + fr = 2`. A useful approach is to provide `ff` as $f_f = 2c_f/(c_f + c_r)$ and `fr` as $f_r = 2c_r/(c_f + c_r)$.
- Refer to the table collection in the distributed load definition.

Abaqus/Standard computes and applies the moving distributed heat fluxes automatically according to the specified Goldak distribution and scanning trajectory.

$$q = q_f \ when \ x \geq 0$$
$$= q_r \ when \ x < 0$$

$$q_{f/r} = \frac{6\sqrt{3} f_{f/r} Q}{abc_{f/r}\pi\sqrt{\pi}} e^{\left(-\frac{3x^2}{c_{f/r}^2}\right)} e^{\left(-\frac{3y^2}{a^2}\right)} e^{\left(-\frac{3z^2}{b^2}\right)}$$

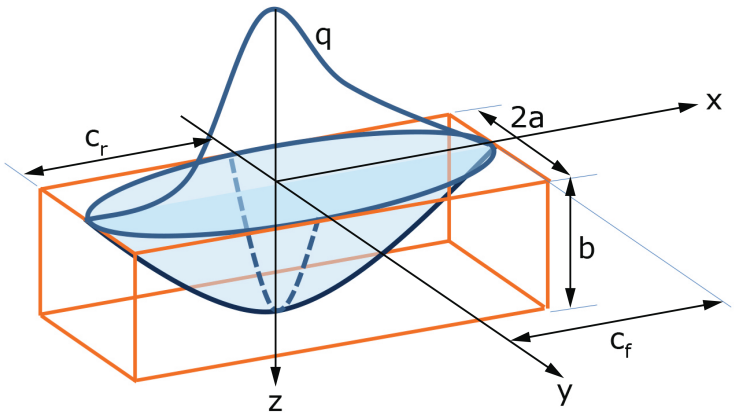


Figure 3: Goldak expression for energy distribution (for the front and rear partitions).

Input File Usage: Use the following options to define a moving heat source with the Goldak distribution:

**EVENT SERIES*, NAME=*event series name*, TYPE=*"ABQ_AM_PowerMagnitude"*
data lines to define the motion of the laser

**TABLE COLLECTION*, NAME="ABQ_AM_*table collection name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MovingHeatSource"*
event series name, "Goldak"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MovingHeatSource_Goldak"*
 # subdivisions in x-direction, *# subdivisions in y-direction*, *# subdivisions in z-direction*,
a, b, c_f, c_r,
f_f, f_r, *box size factor*
**PROPERTY TABLE*, TYPE=*"ABQ_AM_AbsorptionCoeff"*
absorption coefficient
...
**STEP*
**HEAT TRANSFER*
...
**DFLUX*
element set name, MBFNU, ,"ABQ_AM_*table collection name*"
...

Specifying a Moving Heat Source with a Uniform Distribution

If the size of the finite elements used in a thermal analysis is comparable to the size of the laser spot, the laser power can be distributed uniformly over a box-shaped volume. The following steps are required to define the moving heat source completely:

- Define the trajectory of the laser spot in an event series similar to the definition of the concentrated moving heat source.
- Create a table collection with a name that begins with `"ABQ_AM"`. The table collection must contain a parameter table of type *"ABQ_AM_MovingHeatSource"* and a parameter table of type *"ABQ_AM_MovingHeatSource_Uniform"*.
- In the parameter table of type *"ABQ_AM_MovingHeatSource"*, include a reference to the event series for the moving heat source, and set the heat source type to `"Uniform"`.
- In the parameter table of type *"ABQ_AM_MovingHeatSource.Uniform"*, define the parameters of the box-shaped volume.
- Refer to the table collection in the distributed load definition.

Abaqus/Standard computes and applies the moving distributed heat fluxes automatically.

Input File Usage: Use the following options to define a moving heat source with a uniform distribution:

**EVENT SERIES*, NAME=*event series name*, TYPE=*"ABQ_AM_PowerMagnitude"*
data lines to define the motion of the laser
**TABLE COLLECTION*, NAME="ABQ_AM_*table collection name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MovingHeatSource"*
event series name, "Uniform"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MovingHeatSource_Uniform"*
 # subdivisions in x-direction, *# subdivisions in y-direction*, *# subdivisions in z-direction*,
offset in x-direction, *offset in y-direction*, *offset in z-direction*,
box length in x-direction, *box length in y-direction*, *box length in z-direction*,
**PROPERTY TABLE*, TYPE=*"ABQ_AM_AbsorptionCoeff"*
absorption coefficient
...
**STEP*

**HEAT TRANSFER*
...
**DFLUX*
element set name, MBFNU, ,"ABQ_AM_*table collection name*"
...

Specifying Free Surface Radiation and Convective Heat Transfer

For information on specifying free surface radiation and convective heat transfer, see *Progressive Cooling via Convection and Radiation*.

Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Pattern-Based Method

Products: Abaqus/Standard

References:

- *Progressive Element Activation*
- *Sequential thermomechanical analysis of a laser powder bed fusion build*
- *Heat energy balance*
- *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*
- **TABLE COLLECTION*
- **PARAMETER TABLE*
- **PARAMETER TABLE TYPE*
- **PROPERTY TABLE*
- **PROPERTY TABLE TYPE*

Overview

Special-purpose techniques and user subroutines are available to define the relevant process parameters for material deposition and heat sources. You do not need to define a trajectory of a heat source or a recoater movement to perform a sequentially coupled thermal-stress analysis analysis using the pattern-based method.
These internal built-in user subroutines are accessed by starting names for table collections with "ABQ_TMP" as described below.

In a powder bed–type additive manufacturing (AM) process, such as selective laser sintering (SLS) and stereolithography (SLA), a single layer of raw material is deposited by a recoater or a roller blade. Then, a high-powered laser scans a single cross-section of the part following a raster scan pattern over the layer of raw material to fuse it with the previously laid layer underneath. The layer-upon-layer raw material deposition is simulated by progressive element activation in a structural or a thermal analysis, and the laser-induced heating is simulated by a moving heat flux in a thermal analysis.

Specifying Progressive Element Activation and Scan Pattern Parameters

The layer-by-layer deposition of raw material from a recoater or roller blade is simulated using progressive element activation in a structural or a thermal analysis. The following steps are required to define the deposition process and scan pattern parameters completely:

- Create a table collection with a name that begins with "ABQ_TMP". The table collection must contain a parameter table of type *"ABQ_AM_ThermoMech_PatternBased_Activation"* as well as the other parameter tables listed below.
- Define the build parameters (such as slice or layer thickness and build coordinate system) in a parameter table of type *"ABQ_AM_ThermoMech_PatternBased_Activation"*.
- Define the laser power, scan spacing, and speed in a parameter table of type *ABQ_AM_ThermoMech_ScanParameter_Define*.
- Define parameters of rectangular patches that make up a rectangular unit cell (see *Scan Pattern–Mesh Intersection*), which is repeated to cover the cutting plane or a layer, in a parameter table of type *ABQ_AM_ThermoMech_PatternBased_Define*. You can define the following parameters: a local angle, φ;

  the extents of a patch $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$; and a label of the parameter table of type *ABQ_AM_ThermoMech_ScanParameter_Define* defining laser power and other scan parameters to apply to this patch.

- Define the parameters of the bounding box, inside which a particular scan-pattern strategy is active, in a parameter table of type *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define*.

  Alternatively, use a parameter table of type *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset* to assign a particular scan-pattern strategy directly to an element set. Element set–based assignments of scan-pattern strategies have precedence over bounding box–based assignments.

  You can combine both assignment types under the same table collection.
- List the names of all the parameter tables of types *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define* and *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset* defining all active strategies in a table of type *"ABQ_AM_ThermoMech_PatternBased_ScanStrategies"*.
- As described above, you must define at least one parameter table of each of the following types: *"ABQ_AM_ThermoMech_PatternBased_Activation"*, *"ABQ_AM_ThermoMech_PatternBased_Activation"*, *"ABQ_AM_ThermoMech_PatternBased_Define"*, *"ABQ_AM_ThermoMech_PatternBased_ScanStrategies"*, and either *"ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define"* or *"ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset"*.
- Optionally, define parameter tables of types *ABQ_AM_ThermoMech_PatternBased_Advanced* and *ABQ_AM_ThermoMech_Activation_Advanced* to specify advanced options.
- Refer to the table collection when you turn on the progressive element activation feature. You can include only one option to turn on progressive element activation in an analysis.

Abaqus/Standard activates elements automatically according to the specified pattern parameters and automatically estimates the printing time of each layer. Abaqus/Standard assigns a local orientation for anisotropic materials to elements at the time of activation based on the pattern definition. You must define an initial local orientation and refer to it from the section definition for the elements that use an anisotropic material.

A dedicated collection of parameter table, property table, and event series types is available to include all the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage: Use the following options to define progressive element activation and scan parameters in a thermal or a structural analysis:

**TABLE COLLECTION*, NAME="ABQ_TMP_*Activation 1*"
**PARAMETER TABLE*, TYPE="*ABQ_AM_ThermoMech_PatternBased_Activation*"
h, *coordinates defining the build system* $I - J - K$,
recoater time, *print start time*, *TOTALTIME / STEPTIME*
**PARAMETER TABLE*, TYPE="*ABQ_AM_ThermoMech_ScanParameter_Define*",
LABEL = *"Island 1"*
Power, *scan spacing*, *speed*, *on time fraction*
include as many parameter tables of type
ABQ_AM_ThermoMech_ScanParameter_Define as needed
**PARAMETER TABLE*, TYPE="*ABQ_AM_ThermoMech_PatternBased_Define*",
LABEL=*"Pattern 1"*

x_{min}, y_{min}, x_{max}, y_{max},
label of a table of type ABQ_AM_ThermoMech_ScanParameter_Define such as "Island 1", φ
additional data lines to define more patches
**PARAMETER TABLE*,

TYPE="*ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define*",
LABEL=*"Scan region box1"*
label of a table of type ABQ_AM_ThermoMech_PatternBased_Define such as "Pattern 1", θ,
x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}
include as many parameter tables of type
**PARAMETER TABLE*,
TYPE="*ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset*",
LABEL=*"Scan region elset1"*
label of a table of type ABQ_AM_ThermoMech_PatternBased_Define such as "Pattern 1", θ, *element set name*
include as many parameter tables of type ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define as needed
**PARAMETER TABLE*,
TYPE="*ABQ_AM_ThermoMech_PatternBased_ScanStrategies*"
list of labels of parameter tables of types ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define and ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset such as "Scan region box1" and "Scan region elset1" in the analysis
**PARAMETER TABLE*, TYPE="*ABQ_AM_ThermoMech_Activation_Advanced*"
"Full" or "Partial", *min volume fraction threshold for partial activation*, *max volume fraction threshold for partial activation*, *volume fraction threshold for full activation*
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME=*activation name*, ELSET=*element set name*
...
**STEP*
**STATIC* or **HEAT TRANSFER*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*
"ABQ_TMP_*Activation 1*"
...

Specifying a Heat Source

The following steps are required to define the heat source completely that follow the pattern parameters defined above:

- Create a table collection with a name that begins with "ABQ_TMP". The table collection must contain a parameter table of type *ABQ_AM_ThermoMech_PatternParameters* and a property table of type *ABQ_AM_AbsorptionCoeff*.
- In the parameter table, specify the name of the table collection that is referred to when you turn on the progressive element activation feature.
- In the property table, define the absorption coefficient for the material.
- Refer to the table collection in the distributed load definition.

Abaqus/Standard computes and applies moving heat fluxes to each element automatically according to the specified scanning trajectory. There might be more than one distributed load definition. Each distributed load definition must refer to a different table collection. Element sets referred to from distributed load definitions must be subsets of the element set referred to when you specify elements that can be activated during an analysis.

Input File Usage: Use the following options to define a concentrated moving heat flux:

TABLE COLLECTION, NAME="ABQ_TMP_*Laser 1"

Input File Usage:

*PARAMETER TABLE, TYPE="ABQ_AM_ThermoMech_PatternParameters"
"ABQ_TMP_*table collection name" referred from *ACTIVATE ELEMENTS, such as "*ABQ_TMP_*Activation 1"*
*PROPERTY TABLE, TYPE="ABQ_AM_AbsorptionCoeff"
absorption coefficient
...
*STEP
*HEAT TRANSFER
...
*DFLUX
element set name, MBFNU, ,"ABQ_TMP_*Laser 1*"
...

Specifying Free Surface Radiation and Convective Heat Transfer

For information on specifying free surface radiation and convective heat transfer, see *Progressive Cooling via Convection and Radiation*.

Visualization of a Scan Pattern

A scan pattern can be visualized over the part geometry by requesting element solution-dependent field variables for output and plotting them as contours over the finite element mesh. For a pattern-based analysis, the first two element solution-dependent field variables are internally set to the patch ID and scan region ID, respectively (see *Figure 1*).

Input File Usage: Use the following options to visualize a scan pattern:

*SOLID SECTION, ELSET=*element set name*, MATERIAL=*material property name*
*ELEMENT SOLUTION-DEPENDENT VARIABLES
2
1, RegionID
2, Patch/IslandID
…
*OUTPUT, FIELD, FREQUENCY=1
*ELEMENT OUTPUT
ESDV

Thermomechanical Analysis of FDM- and LDED-Type Additive Manufacturing Processes

Products: Abaqus/Standard

References:

- *Progressive Element Activation*
- *Sequential thermomechanical analysis of a directed energy deposition build*
- *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*
- *Accessing Abaqus Table Collections*
- *Accessing Abaqus Event Series Data*
- **EVENT SERIES*
- **EVENT SERIES TYPE*
- **TABLE COLLECTION*
- **PARAMETER TABLE*
- **PARAMETER TABLE TYPE*
- **PROPERTY TABLE*
- **PROPERTY TABLE TYPE*

Overview

Special-purpose techniques and user subroutines are available for FDM- and LDED-type additive manufacturing processes.
These internal built-in user subroutines are accessed by starting names and types with `"ABQ_"` as described below.

In a fusion deposition modeling (FDM)-type additive manufacturing process, the raw material is injected through a nozzle onto a platform. The nozzle traces the cross-section pattern for each layer with the raw material. Materials are typically deposited layer-upon-layer until the build is complete. The raw material can be deposited in a molten state and then hardens as it cools. In some processes, such as laser direct energy deposition (LDED), the raw material is injected in a powdered form and then heated in place by a laser beam.

Specifying Progressive Element Activation

The deposition of raw material from a moving nozzle is simulated using progressive element activation in a structural or a thermal analysis. The cross-section of the nozzle and the bead of the material being deposited are assumed to be rectangular (see *Figure 1*). The following steps are required to define the deposition process completely:

- Define the motion of the nozzle in an event series.
- Create a table collection with a name that begins with `"ABQ_AM"`. The table collection must contain a parameter table of type *ABQ_AM_MaterialDeposition* and a parameter table of type *"ABQ_AM_MaterialDeposition_Bead"*.
- In the parameter table of type *ABQ_AM_MaterialDeposition*, include a reference to the event series for the nozzle motion, and set the deposition process type to `"Bead"`.
- In the parameter table of type *"ABQ_AM_MaterialDeposition_Bead"*, define the process parameters, such as the height and width of the bead.
- Refer to the table collection in the progressive element activation.

Abaqus/Standard activates elements automatically according to the specified nozzle trajectory.

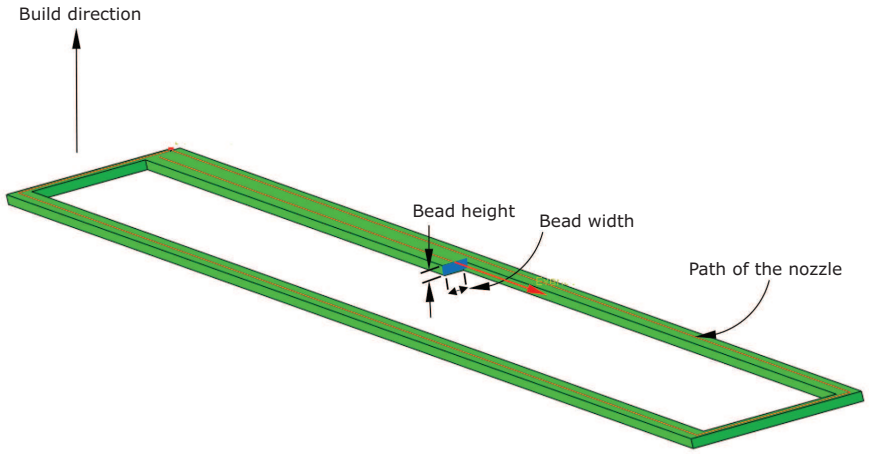


Figure 1: Progressive element activation for FDM and LDED processes.

A dedicated collection of parameter table, property table, and event series types is available to include all of the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all of the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage: Use the following options to define element activation for an FDM- or LDED-type process:

**EVENT SERIES*, NAME=*event series name*, TYPE=*"ABQ_AM_MaterialDeposition"*
data lines to define the motion of the nozzle
**TABLE COLLECTION*, NAME="ABQ_AM.*table_collection_name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition"*
event series name, "Bead"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition_Bead"*
"Z", *bead height*, *bead width*, *activation offset*, "Below"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition_Advanced"*
"Full" or "Partial", *min volume fraction threshold for partial activation*,
max volume fraction threshold for partial activation,
volume fraction threshold for full activation, *update orientation*, *element subdivision order*
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME=*activation name*,
ELSET=*element set name*
...
**STEP*
**STATIC* or **HEAT TRANSFER*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*
"ABQ_AM_*table collection name*"
...

Specifying Progressive Element Activation for a Material Bead with Variable Size and Orientation

You can simulate the deposition of a raw material bead from a moving nozzle using progressive element activation in a structural or a thermal analysis. The height, width, and orientation of the rectangular cross-section of the bead of the deposited raw material can vary with time.

The following steps are required to define the deposition process completely:

- Define an event series of type *"ABQ_AM_MovingHeatSource_5AxisStrategy_VariableCrossSection"*. Define the motion of the nozzle, on/off state of the nozzle, the varying height and width of the rectangular cross-section of the bead, and the local direction vector $\boldsymbol{z}_1$ that defines the bead orientation in the six fields of the event series.
- Create a table collection with a name that begins with "ABQ_AM". The table collection must contain a parameter table of type *"ABQ_AM_MaterialDeposition"*, a parameter table of type *"ABQ_AM_MaterialDeposition_Bead"*, a parameter table of type *"ABQ_AM_MaterialDeposition_5AxisStrategy"* , and a parameter table of type *"ABQ_AM_MaterialDeposition_VariableBeadSize"*.
- In the parameter table of type *"ABQ_AM_MaterialDeposition"*", include a reference to the event series for the nozzle motion, and set the deposition process type to `"Bead"`. The direction defined by the local direction $\boldsymbol{z}_1$ in the event series overrides the stacking direction defined in this parameter table.
- In the parameter table of type *"ABQ_AM_MaterialDeposition_Bead"*, define the process parameters, such as the nominal height and width of the bead.
- In the parameter table of type *"ABQ_AM_MaterialDeposition_VariableBeadSize"*, define the field positions in the event series that define the height and width of the bead.
- In the parameter table of type *"ABQ_AM_MovingHeatSource_5AxisStrategy"*", define the field positions in the event series that define the three components of the local direction vector $\boldsymbol{z}$.
- Optionally, define a parameter table of type *"ABQ_AM_MaterialDeposition_Advanced"* to specify additional settings for the bead.
- Refer to the table collection in the progressive element activation.

A dedicated collection of parameter table, property table, and event series types is available to include all of the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all of the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage: Use the following options to define element activation for an FDM- or LDED-type process:

**EVENT SERIES*, NAME=*event series name*,
TYPE=*"ABQ_AM_MaterialDeposition_5AxisStrategy_VariableCrossSection"*
t_1, x_1, y_1, z_1, 1, $Z1_{1x}$, $Z1_{1y}$, $Z1_{1z}$, h_1, w_1
t_2, x_2, y_2, z_2, 1, $Z1_{2x}$, $Z1_{2y}$, $Z1_{2z}$, h_2, w_2
...
**TABLE COLLECTION*, NAME="ABQ_AM_*table_collection_name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition"*
event series name, "Bead"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialDeposition_Bead"*
`"X"`, `"Y"`, or `"Z"`, *bead height*, *bead width*, *activation offset*, "Below" , "Middle", or "Above"

*PARAMETER TABLE, TYPE="ABQ_AM_MaterialDeposition_5AxisStrategy"
"false", 2, 3, 4
*PARAMETER TABLE, TYPE="ABQ_AM_MaterialDeposition_VariableBeadSize"
5, 6
*PARAMETER TABLE, TYPE="ABQ_AM_MaterialDeposition_Advanced"
"Full" or "Partial", min volume fraction threshold for partial activation,
max volume fraction threshold for partial activation,
volume fraction threshold for full activation, update orientation, element subdivision order
...
*ELEMENT PROGRESSIVE ACTIVATION, NAME=activation name,
ELSET=element set name
...
*STEP
*STATIC or *HEAT TRANSFER
...
*ACTIVATE ELEMENTS, ACTIVATION=activation name
"ABQ_AM_table collection name"
...

Specifying a Moving Heat Source

For information on specifying a moving heat source for FDM- and LDED-type additive manufacturing processes, see *Specifying a Concentrated Moving Heat Source*, *Specifying a Moving Heat Source with a Goldak Distribution*, and *Specifying a Moving Heat Source with a Uniform Distribution*.

The toolpath-mesh intersection module can attach only a single element set to each event series. If you define an event series to activate a bead of material and use the same event series to define the path of a heat source, the heat source applies only to the elements in the bead. To apply the heat source to additional elements, you should define two separate event series, one for the material deposition and one for the moving heat source.

Specifying Free Surface Radiation and Convective Heat Transfer

For information on specifying free surface radiation and convective heat transfer, see *Progressive Cooling via Convection and Radiation*.

Eigenstrain-Based Analysis of Additive Manufacturing Processes

Products: Abaqus/Standard

References:

- *Pattern-based eigenstrain analysis*
- *Trajectory-based eigenstrain analysis*
- *Progressive Element Activation*
- *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*
- *Accessing Abaqus Table Collections*
- *Accessing Abaqus Event Series Data*
- **EVENT SERIES*
- **EVENT SERIES TYPE*
- **TABLE COLLECTION*
- **PARAMETER TABLE*
- **PARAMETER TABLE TYPE*
- **PROPERTY TABLE*
- **PROPERTY TABLE TYPE*

Overview

A special-purpose technique is available for eigenstrain-based simulation of additive manufacturing processes.

This technique employs internal built-in user subroutines that are accessed by using names and types that begin with "ABQ_", as described below.

Abaqus/Standard supports eigenstrain-based analysis of additive manufacturing processes through two approaches: the pattern-based method and the trajectory-based method.

When you use the trajectory-based method, you define the tool path (for example, movement of the laser spot) explicitly using an event series. Abaqus/Standard uses this event series to determine the time of activation of an element and the eigenstrain to apply to the element at the time of its activation. Optionally, you can also include the tool orientation and the dimensions of the deposition bead (height and width) in the event series definition used to specify the tool trajectory.

When you use the pattern-based method, you define a scan pattern that represents an idealized motion of the laser. Abaqus/Standard uses this scan pattern to determine the time of activation of an element and the eigenstrain to assign to that element (see *Scan Pattern–Mesh Intersection*).

Specifying Element Activation and Eigenstrain Using the Pattern-Based Method

To use the pattern-based eigenstrain method, you must complete the following steps: :

- Create a table collection with a name that begins with `"ABQ_EIG"`. The table collection must contain a parameter table of type *"ABQ_AM_EigenStrain_Method"* as well as the other parameter tables listed below.
- In the parameter table of type *"ABQ_AM_EigenStrain_Method"*, set the activation type to `"PatternBased"`.
- Define parameters of rectangular patches that make up a rectangular unit cell (see *Scan Pattern–Mesh Intersection*), which is repeated to cover the cutting plane or a layer, in a parameter table of type *"ABQ_AM_EigenStrain_PatternBased_Define"*. You can define the following parameters: a local angle, φ;

the extents of a patch (x_{min}, y_{min}) and (x_{max}, y_{max}); and a label of the parameter table defining an eigenstrain to apply to this patch.
- Define the six eigenstrain components in a parameter table of type *"ABQ_AM_EigenStrain_Define"*.
- Define the parameters of the bounding box, inside which a particular scan-pattern strategy is active, in a parameter table of type *"ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define"*.

  Alternatively, use a parameter table of type *"ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset"* to assign a particular scan-pattern strategy directly to an element set. Element set–based assignments of scan-pattern strategies have precedence over bounding box–based assignments.

  You can combine both assignment types under the same table collection.
- Define the build parameters (such as slice or layer thickness and build coordinate system) in a parameter table of type *"ABQ_AM_EigenStrain_PatternBased_Activation"*.
- Optionally, include a parameter table of type *"ABQ_AM_EigenStrain_Activation_Advanced"* to specify advanced options.
- Refer to the table collection in the progressive element activation.

A dedicated collection of parameter table, property table, and event series types is available to include all the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage: Use the following options to define scan pattern parameters and eigenstrains:

**TABLE COLLECTION*, NAME="ABQ_EIG.*user-defined name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_Method"*
"PatternBased"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_PatternBased_Activation"*
h, *coordinates defining the build system* $I - J - K$
"Total Time", *print start time*, *print end time*
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_PatternBased_Advanced"*
"LAYERBYLAYER" or "SWEEP", True or False
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_PatternBased_ScanStrategies"*
list of labels of parameter tables of type
"ABQ_AM_*EigenStrain_PatternBased_ScanStrategy_Define" or type*
*"*ABQ_AM_*EigenStrain_PatternBased_ScanStrategy_Define_Elset" in analysis*
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define"*,
LABEL="ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Box1"
"l*abel of a table of type* ABQ_AM_*EigenStrain_PatternBased_Define*", θ,
x_{min}, y_{min}, z_{min},x_{max}, y_{max}, z_{max}
include as many parameter tables of type
"ABQ_AM_*EigenStrain_PatternBased_ScanStrategy_Define" as needed*
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset"*,
LABEL="ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Elset1"
"label of a table of type ABQ_AM_*EigenStrain_PatternBased_Define*", θ, *element set name*

include as many parameter tables of type
"ABQ_AM_*EigenStrain_PatternBased_ScanStrategy_Define_Elset" as needed*
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_PatternBased_Define"*,
LABEL=*user-defined label*

x_{min}, y_{min}, x_{max}, y_{max}, *label of a table of type "*ABQ_AM_*EigenStrain_Define"*,
φ
additional data lines to define more patches
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_Define"*,
LABEL=*user-defined label*

ε^*_{11}, ε^*_{22}, ε^*_{33}, ε^*_{12}, ε^*_{13}, ε^*_{23}
include as many parameter tables of type "ABQ_AM_EigenStrain_Define" as needed
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME="EigenstrainActivation",
ELSET=printElements
...
**STEP*
**STATIC*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*
"ABQ_EIG.*user-defined name*"
...

Visualization of a Scan Pattern

A scan pattern can be visualized over the part geometry by requesting element solution-dependent field variables for output and plotting them as contours over the finite element mesh. For a pattern-based eigenstrain analysis, the first two element solution-dependent field variables are internally set to the patch ID and scan region ID, respectively (see *Figure 1*). For a trajectory-based eigenstrain analysis, the first element solution-dependent field variable is internally set to the rule ID.

Input File Usage: Use the following options to visualize a scan pattern:

**SOLID SECTION*, ELSET=*element set name*, MATERIAL=*material property name*
**ELEMENT SOLUTION-DEPENDENT VARIABLES*
2
1, RegionID,
2, Patch/IslandID,
…
**OUTPUT*, FIELD, FREQUENCY=*1*
**ELEMENT OUTPUT*
ESDV

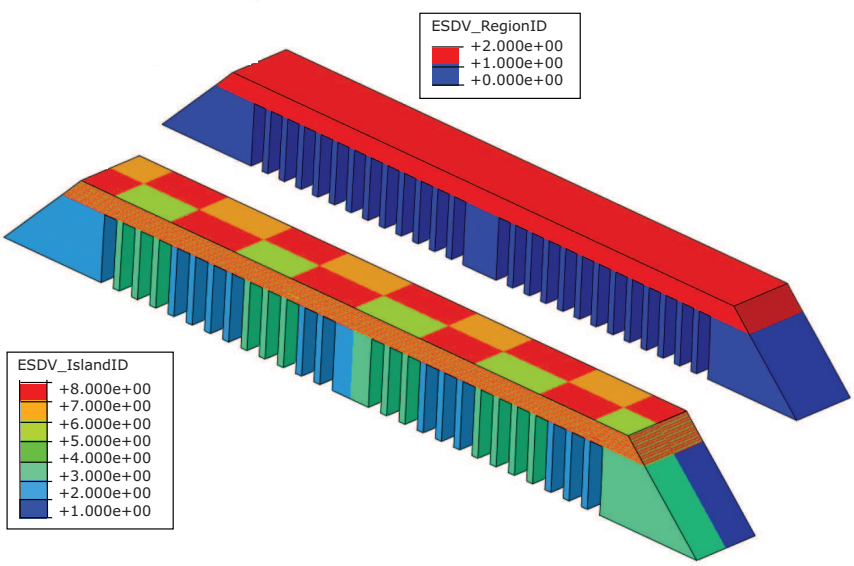


Figure 1: Quilt-style contour plots of scan pattern island ID and scanning region ID.

Specifying Element Activation and Eigenstrain Using the Trajectory-Based Method

To use the trajectory-based eigenstrain method, you must complete the following steps:

- Create a table collection with a name that begins with `"ABQ_EIG"`.

  The table collection must contain a parameter table of type *"ABQ_AM_EigenStrain_Method"* as well as the other parameter tables listed below.
- In the parameter table of type *ABQ_AM_EigenStrain_Method"*, set the activation type to `"TrajectoryBased"`.
- Use a parameter table of type *"ABQ_AM_EigenStrain_TrajectoryBased_Activation"* to define the parameters of the rectangular cross-section of the activation front (see *Figure 1* in *Thermomechanical Analysis of FDM- and LDED-Type Additive Manufacturing Processes*, and *Figure 2*), and to specify the name of the event series defining the tool path and the status of the active strain definitions at locations along the tool trajectory.

  The event series can have one of the following types depending on the content to be provided:

Event Series Type	Use to Define
"ABQ_AM_HeatSourceTrajectory_RuleID"	The tool location and active strain definition as a function of time
"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"	The tool location and active strain definition as a function of time and a time-dependent tool orientation
"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection"	The tool location and active strain definition as a function of time, a time-dependent tool orientation, and time-dependent bead cross-section dimensions

  All event series types allow you to change the active eigenstrain definition along the tool path, if needed. Using "0" for the active eigenstrain definition rule has a special meaning and corresponds to turning the tool off. When the tool is active (that is, it has a nonzero eigenstrain rule ID), the tool direction cannot be parallel to the event series segment.
- Define the six eigenstrain components in a parameter table of type *"ABQ_AM_EigenStrain_Define"*.

You can include multiple parameter tables of this type in the table collection, each defining a separate set of components of the eigenstrain tensor. You must assign a unique label to each table.

- Define a scan rule by including a parameter table of type *"ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define"* that refers to a parameter table defining an eigenstrain.

  You must assign a unique label to the scan rule, and you must list all the scan rules defined in the table collection in a parameter table of type *"ABQ_AM_EigenStrain_TrajectoryBased_Rules"*.

  The position index (starting from one) of each rule in this list is the rule ID number that is specified as the first field in the event series of types *"ABQ_AM_HeatSourceTrajectory_RuleID"*, *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"*, or *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection"*.
- Optionally, include a parameter table of type *"ABQ_AM_EigenStrain_Activation_Advanced"* to specify advanced options.
- If you need to use a time-dependent tool orientation, you can include a parameter table of type *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"*.

  This parameter table must provide the IDs of the fields defining the tool orientation vector in the event series of type *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"* or type *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection"*.
- You can include a parameter table of type *"ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize"*, if time-dependent bead dimensions (height and width) are provided in the event series of type *"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection"* that defines the tool trajectory.

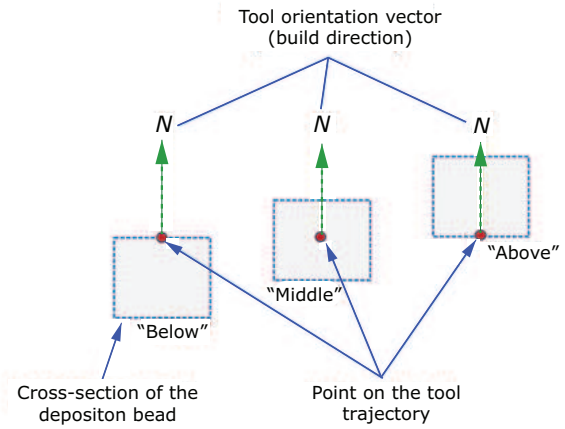


Figure 2: Position of the deposition bead relative to the tool path.

Abaqus/Standard assumes the eigenstrain components to be defined with respect to a local (orthonormal) coordinate system $\boldsymbol{E}_1^{loc}, \boldsymbol{E}_2^{loc}, \boldsymbol{E}_3^{loc}$, as shown in *Figure 3*.

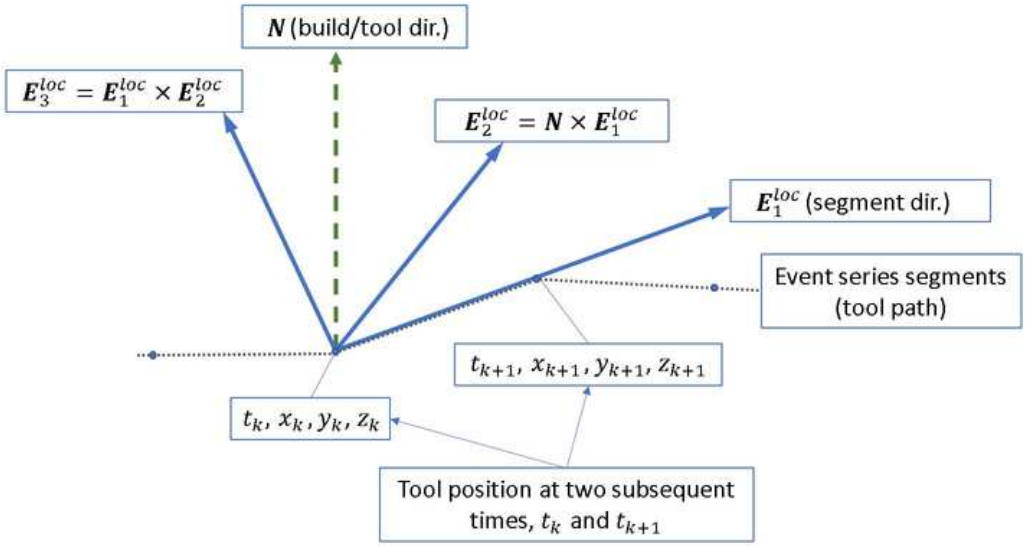


Figure 3: Local orientation along the tool trajectory when activating solid elements.

For solid elements the local coordinate system follows the tool trajectory and depends on the tool (build) orientation, such that

- $\boldsymbol{E}_1^{loc}$ is along the current tool trajectory segment,
- $\boldsymbol{E}_2^{loc}$ is computed based on the cross-product between the current tool direction vector, $\boldsymbol{N}$, and $\boldsymbol{E}_1^{loc}$ (that is, $\boldsymbol{E}_2^{loc} = \boldsymbol{N} \times \boldsymbol{E}_1^{loc}$), and
- $\boldsymbol{E}_3^{loc}$ is computed using $\boldsymbol{E}_3^{loc} = \boldsymbol{E}_1^{loc} \times \boldsymbol{E}_2^{loc}$.

For shell elements, a different local orientation $\boldsymbol{E}_1^{loc,sh}, \boldsymbol{E}_2^{loc,sh}, \boldsymbol{E}_3^{loc,sh}$ is computed that depends on the average shell element normal, $\boldsymbol{N}_{shell}^{avg}$ (and $\boldsymbol{E}_3^{loc}$), such that

$$\begin{aligned} \boldsymbol{E}_3^{loc,sh} &= \boldsymbol{N}_{shell}^{avg}, \\ \boldsymbol{E}_1^{loc,sh} &= \boldsymbol{N}_{shell}^{avg} \times \boldsymbol{E}_3^{loc}, \\ \boldsymbol{E}_2^{loc,sh} &= \boldsymbol{N}_{shell}^{avg} \times \boldsymbol{E}_1^{loc,sh}. \end{aligned}$$

For solid elements, if the material assigned to the elements to be activated includes an initial orientation, Abaqus/Standard updates that orientation with the local orientation shown in *Figure 3* at the moment of element activation; otherwise, Abaqus/Standard transforms the eigenstrain to the global coordinate directions. For shell elements, Abaqus/Standard cannot update the orientation; instead, Abaqus/Standard transforms the eigenstrain to the local shell orientation.

Input File Usage: Use the following options to define trajectory-based activation in an eigenstrain-based analysis:

**EVENT SERIES*, NAME=*event series name*,
TYPE=*"ABQ_AM_HeatSourceTrajectory_RuleID"*
data lines to define the motion of the nozzle and eigenstrain rule id at locations along the tool trajectory
**EVENT SERIES*, NAME=*event series name*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"*

data lines to define the motion of the nozzle, eigenstrain rule id and tool direction at locations along the tool trajectory
**EVENT SERIES*, NAME=*event series name*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection"*
data lines to define the motion of the nozzle, eigenstrain rule id, tool direction and bead height and width at locations along the tool trajectory
**TABLE COLLECTION*, NAME="ABQ_EIG.*user-defined name*"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_Method"*
"TrajectoryBased",
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_Rules"*
list of labels of parameter tables of type "ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define" in analysis
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define"*,
LABEL=*user-defined label*
label of a parameter table of type "ABQ_AM_EigenStrain_Define"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_EigenStrain_Define"*,
LABEL=*user-defined label*

$\varepsilon^*_{11}, \varepsilon^*_{22}, \varepsilon^*_{33}, \varepsilon^*_{12}, \varepsilon^*_{13}, \varepsilon^*_{23}$
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_Activation"*
event series name, stack direction, bead height, bead width, deposition position
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"*
"True", 2, 3, 4
**PARAMETER TABLE*,
TYPE=*"ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize"*
5 , 6
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME="EigenstrainActivation",
ELSET=printElements
...
**STEP*
**STATIC*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*
"ABQ_EIG.*user-defined name*"
...

Limitations

The eigenstrain-based simulation method for additive manufacturing does not support material models with hyperelastic, hyperfoam, or low-density foam constitutive responses. If elements assigned any of these material behaviors are activated during the analysis, Abaqus/Standard does not apply the eigenstrains defined in the model to those elements.

Machining and Material Removal Process

Products: Abaqus/Standard

References:

- *Pattern-based eigenstrain analysis*
- *Progressive Element Activation*
- *Defining Moving or Stationary Nonuniform Heat Flux in User Subroutine UMDFLUX*
- *Accessing Abaqus Table Collections*
- *Accessing Abaqus Event Series Data*
- **EVENT SERIES*
- **EVENT SERIES TYPE*
- **TABLE COLLECTION*
- **PARAMETER TABLE*
- **PARAMETER TABLE TYPE*
- **PROPERTY TABLE*
- **PROPERTY TABLE TYPE*

Overview

Special-purpose techniques and user subroutines are available for machining and material removal processes, which often follow an additive manufacturing process.
These internal built-in user subroutines are accessed by starting names and types with `"ABQ_"` as described below.

In a machining process or a cutting operation that often follows a powder-bed type or a fusion deposition modeling (FDM)-type additive manufacturing process, the material is removed either by an electrical discharge machining (EDM) operation or by a cutting tool. A toolpath defines a path of the operation in space that describes how the material is removed over time.

Specifying Progressive Element Activation for Material Removal

The removal of material is simulated using progressive element activation in a structural or a thermal analysis. The cross-section of the bead of the material being removed is assumed to be rectangular. The following steps are required to define the removal process completely:

- Define the motion of the nozzle in an event series. The event series must have a least one field that defines the “on/off” state of the tool. Optionally, the event series can also define the varying height and width of the cutting operation, and the local direction vector $\boldsymbol{z}_1$, that defines the orientation of the tool, in five additional fields.
- Create a table collection with a name that begins with "ABQ_AM". The table collection must contain parameter tables of types *"ABQ_AM_MaterialRemoval"* and *"ABQ_AM_MaterialRemoval_Bead"*.
- In the parameter table of type *"ABQ_AM_MaterialRemoval"*, include a reference to the event series for the cutting tool motion.
- In the parameter table of type *"ABQ_AM_MaterialRemoval_Bead"*, define the process parameters, such as the height and width of the bead. If the height and the width are both set to zero, Abaqus/Standard assumes the height to be equal to the average height of a finite element in the element set referenced from the progressive element activation and the width is large enough to cover the entire element set.

- Optionally, add a parameter table of type *"ABQ_AM_MaterialRemoval_VariableBeadSize"* to define the field positions in the event series that define the height and the width of the material removal tool's cross-section.
- Optionally, add a parameter table of type *"ABQ_AM_MaterialRemoval_5AxisStrategy"* to define the field positions in the event series that define the three components of the local direction vector $\boldsymbol{z_1}$ of the tool.
- Optionally, define a parameter table of type *"ABQ_AM_MaterialRemoval_Advanced"*, to specify advanced settings for the element deactivation process.
- Refer to the table collection in the progressive element activation.

A dedicated collection of parameter table, property table, and event series types is available to include all of the definitions required by special-purpose techniques for additive manufacturing. These special-purpose parameter table, property table, and event series type definitions are generated internally. You need to explicitly include them in your model only if you output the results of your additive manufacturing simulation in SIM format. If needed, you can use the **abaqus fetch** utility to obtain the file containing all of the type definitions of parameter tables, property tables, and event series required by the special-purpose techniques for additive manufacturing as follows:

```
abaqus fetch job=ABQ_am_special_purpose_types.inp
```

Input File Usage:

Use the following options to define element removal (in this example, a user-defined event series type is used `MaterialRemoval_5AxisStrategy`):

**EVENT SERIES TYPE*, FIELDS=4, NAME=MaterialRemoval_5AxisStrategy
"On and Off", Dimensionless,
"ES field ID for normal vector componet x", L
"ES field ID for normal vector componet y", L
"ES field ID for normal vector componet z", L
**EVENT SERIES*, NAME=*event series name*,
TYPE=`MaterialRemoval_5AxisStrategy`
t_1, x_1, y_1, z_1, 1, $Z1_{1x}$, $Z1_{1y}$, $Z1_{1z}$
t_2, x_2, y_2, z_2, 1, $Z1_{2x}$, $Z1_{2y}$, $Z1_{2z}$
...
**TABLE COLLECTION*, NAME="ABQ_AM_*table_collection_name*"
**PARAMETER TABLE*, TYPE= *"ABQ_AM_MaterialRemoval"*
event series name
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialRemoval_Bead"*
`"X"`, `"Y"`, or `"Z"`, *bead height*, *bead width*, *activation offset*, "Below" , "Middle", or "Above"
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialRemoval_5AxisStrategy"*
"True" or "False", *2*, *3*, *4*
**PARAMETER TABLE*, TYPE=*"ABQ_AM_MaterialRemoval_Advanced"*
"Full" or "Partial", *min volume fraction threshold for partial deactivation*,
max volume fraction threshold for partial deactivation,
volume fraction threshold for full deactivation, *update orientation*, *element subdivision order*
...
**ELEMENT PROGRESSIVE ACTIVATION*, NAME=*activation name*,
ELSET=*element set name*
...
**STEP*
**STATIC* or **HEAT TRANSFER*
...
**ACTIVATE ELEMENTS*, ACTIVATION=*activation name*

"ABQ_AM_*table collection name*"
...

These parameter table types are used in special-purpose techniques for additive manufacturing and for modeling metallurgical phase transformations.

In this section:

- *ABQ_AM_BladeInterference*
- *ABQ_AM_EigenStrain_Activation_Advanced*
- *ABQ_AM_EigenStrain_Define*
- *ABQ_AM_EigenStrain_Method*
- *ABQ_AM_EigenStrain_PatternBased_Activation*
- *ABQ_AM_EigenStrain_PatternBased_Advanced*
- *ABQ_AM_EigenStrain_PatternBased_Define*
- *ABQ_AM_EigenStrain_PatternBased_ScanStrategies*
- *ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define*
- *ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset*
- *ABQ_AM_EigenStrain_TrajectoryBased_Activation*
- *ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define*
- *ABQ_AM_EigenStrain_TrajectoryBased_Rules*
- *ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy*
- *ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize*
- *ABQ_AM_MaterialDeposition*
- *ABQ_AM_MaterialDeposition_Advanced*
- *ABQ_AM_MaterialDeposition_Bead*
- *ABQ_AM_MaterialDeposition_Bead_Orientation*
- *ABQ_AM_MaterialDeposition_5AxisStrategy*
- *ABQ_AM_MaterialDeposition_VariableBeadSize*
- *ABQ_AM_MaterialRemoval*
- *ABQ_AM_MaterialRemoval_5AxisStrategy*
- *ABQ_AM_MaterialRemoval_Advanced*
- *ABQ_AM_MaterialRemoval_Bead*
- *ABQ_AM_MaterialRemoval_Bead_Orientation*
- *ABQ_AM_MaterialRemoval_VariableBeadSize*
- *ABQ_AM_MovingHeatSource*
- *ABQ_AM_MovingHeatSource_5AxisStrategy*
- *ABQ_AM_MovingHeatSource_Advanced*
- *ABQ_AM_MovingHeatSource_Goldak*
- *ABQ_AM_MovingHeatSource_Goldak_Position*
- *ABQ_AM_MovingHeatSource_Uniform*
- *ABQ_AM_MovingHeatSource_VariableBeadSize*
- *ABQ_AM_ThermoMech_Activation_Advanced*
- *ABQ_AM_ThermoMech_PatternBased_Activation*
- *ABQ_AM_ThermoMech_PatternBased_Advanced*
- *ABQ_AM_ThermoMech_PatternBased_Define*
- *ABQ_AM_ThermoMech_PatternBased_ScanStrategies*

- *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define*
- *ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset*
- *ABQ_AM_ThermoMech_PatternParameters*
- *ABQ_AM_ThermoMech_ScanParameter_Define*

ABQ_AM_BladeInterference

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

This table is used for blade interference output. When requesting this output, you must include a parameter table of type `"ABQ_AM_BladeInterference"` in the table collection that you specify to activate elements in the step.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_BladeInterference", PARAMETERS=2
FLOAT, , "Powder Thickness", , ,
FLOAT, , "Risk Factor", , ,
```

Parameters

Powder Thickness	Thickness of the powder layer. If the value is zero, the thickness comes from the toolpath-mesh intersection module; otherwise, this value is used.
Risk Factor	Risk factor representing the powder blade flexibility. A value of zero indicates that the blade cannot move above the powder layer thickness. A value of one indicates that the blade can move up a distance equal to the powder layer thickness.

ABQ_AM_EigenStrain_Activation_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_EigenStrain_Activation_Advanced"`. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_Activation_Advanced",
PARAMETERS=5
 STRING, "Full", "Element Activation", , "Full|Partial",
 FLOAT, 0.0, "Min Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 1.0, "Max Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 0.0, "Volume Fraction Threshold for Full Activation", , ,
 STRING, "FirstIntersection", "Averaging Technology", ,
"FirstIntersection|Averaged",
```

Parameters

Element Activation
Set to `"Full"` to indicate that elements are activated and their volume fractions set to one. Set to `"Partial"` to indicate that elements are activated when material deposits inside the element and the volume fractions progressively increase from the minimum volume fraction threshold to one as more material deposits. Default value is `"Full"`.

Minimum Volume Fraction Threshold for Partial Activation
For activation type `"Partial"`, elements are considered to have at least the minimum threshold volume fraction when any amount of material is deposited inside the element. The minimum volume fraction threshold should be set to a value between zero and one.

Default value is zero.

Maximum Volume Fraction Threshold for Partial Activation
For activation type `"Partial"`, the maximum volume fraction threshold indicates a volume fraction above which element volume fractions are considered to be one.

Default value is one.

Volume Fraction Threshold for Full Activation
For activation type `"Full"`, the volume fraction threshold indicates a volume fraction below which elements are kept inactive and above which elements are activated with a volume fraction of one.

Default value is zero.

Averaging Technology
Unused.

ABQ_AM_EigenStrain_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_EigenStrain_Define"` in the table collection. Optionally, you can include multiple tables of this type. Each table must have a unique label. Only one set of data is allowed in each table.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_Define", PARAMETERS=6
 FLOAT, 0.0, "Eig11", , ,
 FLOAT, 0.0, "Eig22", , ,
 FLOAT, 0.0, "Eig33", , ,
 FLOAT, 0.0, "Eig12", , ,
 FLOAT, 0.0, "Eig13", , ,
 FLOAT, 0.0, "Eig23", , ,
```

Parameters

Eig11	11 component of the eigenstrain tensor.
Eig22	22 component of the eigenstrain tensor.
Eig33	33 component of the eigenstrain tensor.
Eig12	12 component of the eigenstrain tensor.
Eig13	13 component of the eigenstrain tensor.
Eig23	23 component of the eigenstrain tensor.

ABQ_AM_EigenStrain_Method

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type "ABQ_AM_EigenStrain_Method" in the table collection. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_Method", PARAMETERS=1
 STRING, "TrajectoryBased", "Activation Type", ,
"PatternBased|TrajectoryBased",
```

Parameters

Activation Type Set to "PatternBased" to indicate a scan pattern based on an eigenstrain analysis. Set to "TrajectoryBased" to indicate a trajectory-based eigenstrain analysis.

ABQ_AM_EigenStrain_PatternBased_Activation

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_EigenStrain_PatternBased_Activation"` in the table collection. Only one set of data must be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_PatternBased_Activation",
PARAMETERS=13
 FLOAT, 1.0, "Slice Height", , ,
 FLOAT, 1.0, "OriAX", , ,
 FLOAT, 0.0, "OriAY", , ,
 FLOAT, 0.0, "OriAZ", , ,
 FLOAT, 0.0, "OriBX", , ,
 FLOAT, 1.0, "OriBY", , ,
 FLOAT, 0.0, "OriBZ", , ,
 FLOAT, 0.0, "OriCX", , ,
 FLOAT, 0.0, "OriCY", , ,
 FLOAT, 0.0, "OriCZ", , ,
 STRING, "TotalTime", "Total/Step Time", , "TotalTime|StepTime",
 FLOAT, -1e36, "Start Time", , ,
 FLOAT, +1e36, "End Time", , ,
```

Parameters

For reference, see *Figure 1*.

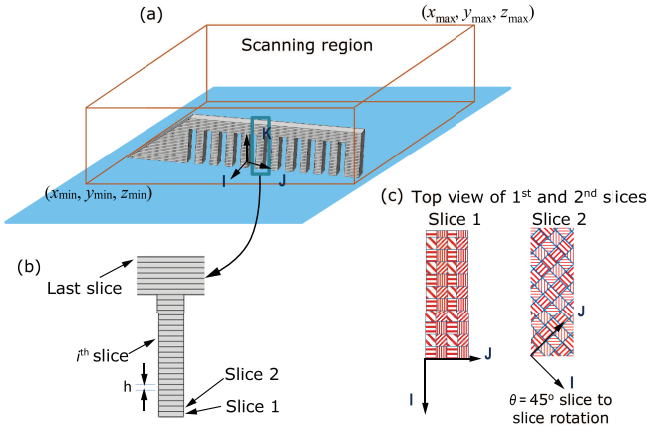


Figure 1: A scan pattern with four pattern patches.

Slice Height	Height of slice for scan pattern.
OriAX, OriAY, OriAZ, OriBX, OriBY, OriBZ, OriCX, OriCY, OriCZ	*X*-, *Y*-, and *Z*-coordinates of points *a*, *b*, and *c* for defining the orientation of the build system *I*–*J*–*K* and its origin. Point *c* is the origin of the system, point *a* must lie on the *I*-axis, and point *b* must lie on the *I*–*J* plane. Although not necessary, it is intuitive to select point *b* such that it is on or near the local *J*-axis.
Total/Step Time	Set to `"TotalTime"` to indicate that the print start and end times are in total time. Set to `"StepTime"` to indicate that the print start and end times are in step time.
Start Time, End Time	Print start and end times.

ABQ_AM_EigenStrain_PatternBased_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_EigenStrain_PatternBased_Advanced"` in the table collection. The table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_PatternBased_Advanced",
PARAMETERS=2
 STRING, "SWEEP", "Activation Technique", , "SWEEP|LAYERBYLAYER",
 STRING, "TRUE", "One Slice per Time Increment", , "TRUE|FALSE",
```

Parameters

Activation Technique	Set to `"LAYERBYLAYER"` to activate all elements of a layer/slice simultaneously. Set to `"SWEEP"` for progression of the element activation front in the *I*-direction in a layer/slice.
One Slice per Time Increment	Set to `"TRUE"` to control the time increment size automatically such that an increment always ends when all elements of a slice/layer are activated.

ABQ_AM_EigenStrain_PatternBased_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type "ABQ_AM_EigenStrain_PatternBased_Define" in the table collection. Each table must have a unique label. Each table can have multiple sets of data; each set defines one patch/island in a scan pattern.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_PatternBased_Define",
PARAMETERS=6
 FLOAT, 0.0, "x_min", , ,
 FLOAT, 0.0, "y_min", , ,
 FLOAT, 1000.0, "x_max", , ,
 FLOAT, 1000.0, "y_max", , ,
 STRING, " ", Label of an ABQ_AM_EigenStrain_Define Table", , ,
 FLOAT, 0.0, "Local In-plane Rotation (in degrees)", , ,
```

Parameters

For reference, see *Figure 1*.

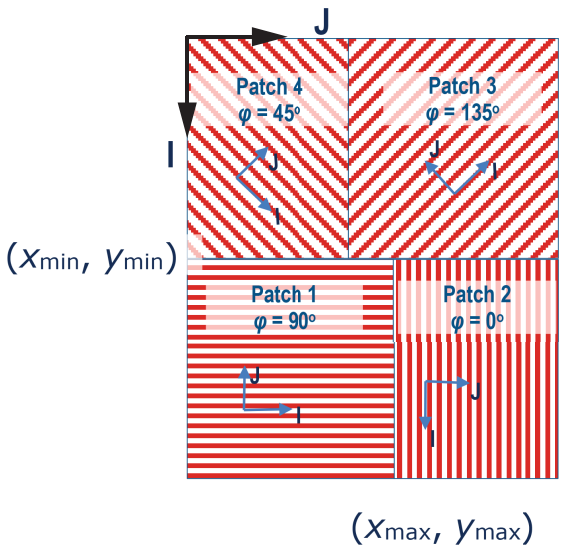


Figure 1: A scan pattern with four patches.

x_{min}, y_{min}, x_{max}, y_{max}	Extent of a scan patch.

Label of an `ABQ_AM_EigenStrain_Define` Table	Refers to the eigenstrain tensor that will be applied to this pattern patch.
Local In-plane Rotation	Local in-plane rotation, φ, of the patch (in degrees).

ABQ_AM_EigenStrain_PatternBased_ScanStrategies

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_EigenStrain_PatternBased_ScanStrategies"` in the table collection. The table can define multiple sets of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_PatternBased_ScanStrategies",
 PARAMETERS=1
 STRING, " ", "Scan Strategy", , ,
```

Parameters

Scan Strategy	Label of a table of type `ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define`. Sets of data from this table define active strategies in the analysis.

ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type `ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define` in the table collection. Each table must have a unique label. Each table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define", PARAMETERS=8
 STRING, " ", "Pattern", , ,
 FLOAT, 0.0, "Relative Rotation Angle (in degrees)", , ,
 FLOAT, -1e36, "Xmin", , ,
 FLOAT, -1e36, "Ymin", , ,
 FLOAT, -1e36, "Zmin", , ,
 FLOAT, 1e36, "Xmax", , ,
 FLOAT, 1e36, "Ymax", , ,
 FLOAT, 1e36, "Zmax", , ,
```

Parameters

For reference, see *Figure 1*.

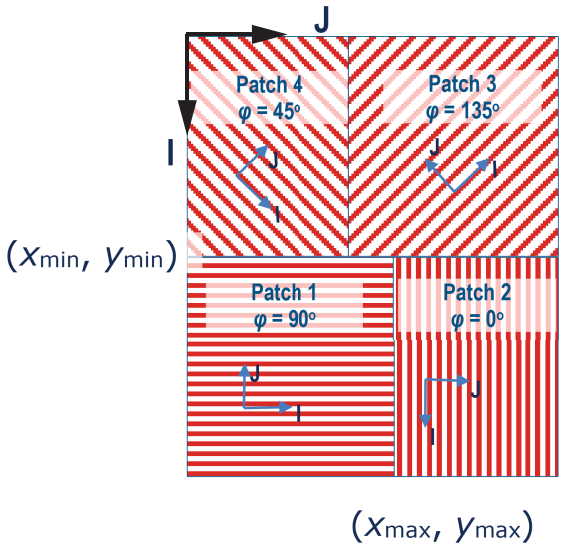


Figure 1: A scan pattern with four patches.

Pattern	Label of a table of type `ABQ_AM_EigenStrain_PatternBased_Define`.
Relative Rotation Angle	Layer-to-layer (slice-to-slice) rotation angle, Θ.
$\mathbf{X_{min}}$, $\mathbf{Y_{min}}$, $\mathbf{Z_{min}}$, $\mathbf{X_{max}}$, $\mathbf{Y_{max}}$, $\mathbf{Z_{max}}$	Extent of a scanning region.

ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type `ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset` in the table collection. Each table must have a unique label. Each table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_EigenStrain_PatternBased_ScanStrategy_Define_Elset",
PARAMETERS=3
 STRING, " ", "Pattern", , ,
 FLOAT,  0.0, "Relative Rotation Angle (in degrees)", , ,
 STRING, " ", "Element Set Name", , ,
```

Parameters

For reference, see *Figure 1*.

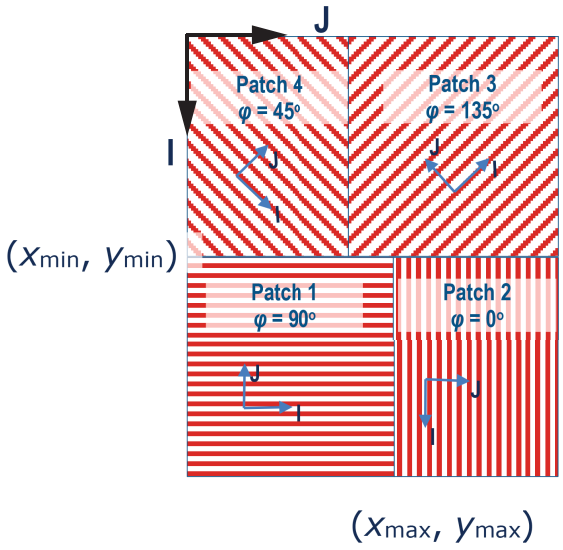


Figure 1: A scan pattern with four patches.

Pattern	Label of a table of type `ABQ_AM_EigenStrain_PatternBased_Define`.
Relative Rotation Angle	Layer-to-layer (slice-to-slice) rotation angle, θ.

Element Set Name Name of the element set containing the elements qualified to be assigned the scan pattern.

ABQ_AM_EigenStrain_TrajectoryBased_Activation

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `ABQ_AM_EigenStrain_TrajectoryBased_Activation` in the table collection. Only one set of data must be defined.

Tables of type `ABQ_AM_MaterialDeposition"`, `ABQ_AM_MovingHeatSource`, and `ABQ_AM_EigenStrain_TrajectoryBased_Activation` cannot refer to the same event series in an analysis.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,NAME="ABQ_AM_EigenStrain_TrajectoryBased_Activation",
 PARAMETERS=5
 STRING, " ", "Deposition Head Event Series", , ,
 STRING, "Z", "Stack Direction", , "X|Y|Z",
 FLOAT, "1.0", "Bead Height", , ,
 FLOAT, "1.0", "Bead Width", , ,
 STRING, "Below", "Deposition Position", , "Below|Middle|Above",
```

Parameters

Deposition Head Event Series Name of an event series of type `ABQ_AM_HeatSourceTrajectory_RuleID`, type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy`, or type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection` describing the trajectory of the toolpath.

Stack Direction Direction of progressive material deposition layers. Options are `X`, `Y`, and `Z` indicating the global x_g-, y_g-, or z_g-direction, respectively.

If you provide time-dependent directions, this direction is replaced by the event series of type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy` or type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection`.

Bead Height Height (measured parallel to the stack/tool direction) of the deposited bead of material.

If the model considers time-dependent bead dimensions, this value is replaced with the values specified in the event series of type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection`.

Bead Width Width (measured perpendicular to the stack/tool direction) of the deposited bead of material.

If the model considers time-dependent bead dimensions, this value is replaced with the values specified in the event series of type `ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection`.

Deposition Position Set to `Above`, `Middle`, or `Below` to indicate that the deposited material bead is situated above, through, or below the path in the stack/tool direction.

ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type "ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define" in the table collection. Only one set of data must be defined. Each table must have a unique label.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define",
 PARAMETERS=1
 STRING, " ", "Label of an ABQ_AM_EigenStrain_Define Table" , , "
```

Parameters

Label of an `ABQ_AM_EigenStrain_Define` Table	Label of a table of type `ABQ_AM_EigenStrain_Define`.

ABQ_AM_EigenStrain_TrajectoryBased_Rules

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_EigenStrain_TrajectoryBased_Rules"` in the table collection. Multiple sets of data can be defined. An integer rule ID that starts from one is assigned internally to the sets of data defined in this table. You must reference this rule ID from the event series that defines the trajectory of the toolpath. See event series type `ABQ_AM_HeatSourceTrajectory_RuleID`.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_EigenStrain_TrajectoryBased_Rules",
PARAMETERS=1
 STRING, " ", "Trajectory Rule Name", , ,
```

Parameters

Trajectory Rule Name Label of a table of type `ABQ_AM_EigenStrain_TrajectoryBased_Rule_Define`. Multiple `Trajectory Rule Name` parameters can be defined. The list of these rule names constitutes the active rules in the analysis.

ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy

Products: Abaqus/Standard

References:

- *Eigenstrain-Based Analysis of Additive Manufacturing Processes*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used to specify the tool orientation at locations along an arbitrary curve in space (defined using an event series) in a trajectory-based eigenstrain additive manufacturing simulation.

For trajectory-based eigenstrain simulations of additive manufacturing processes, if your analysis requires a time-dependent tool orientation (for example, 5-axis deposition), you must include a parameter table of type `"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"` in the table collection used for element activation to specify the tool orientation at the locations along the tool trajectory.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy", PARAMETERS=4
 STRING, "True", "Ignore Lead Angle/Consider Only Tilt", , "True|False"
 INTEGER, 2, "Event Series Field ID for X-Component of Local Direction
Vector Z1", " "
 INTEGER, 3, "Event Series Field ID for Y-Component of Local Direction
Vector Z1", " "
 INTEGER, 4, "Event Series Field ID for Z-Component of Local Direction
Vector Z1", " "
```

Parameters

Ignore Lead Angle/Consider Only Tilt	If set to `"True"`, Abaqus/Standard considers only the component of the normal vector, $\boldsymbol{n}$, which is perpendicular to the toolpath trajectory. Set to `"True"` if only an approximation of local direction vector $\boldsymbol{z_1}$ is defined in the event series. The default value is `"True"`.
Event Series Field ID for X-Component of Local Direction Vector Z1	Set to 2 by default, indicating that the second field in the event series defines the x-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Y-Component of Local Direction Vector Z1	Set to 3 by default, indicating that the third field in the event series defines the y-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Z-Component of Local Direction Vector Z1	Set to 4 by default, indicating that the fourth field in the event series defines the z-component of local direction vector $\boldsymbol{z_1}$.

ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize

Products: Abaqus/Standard

References:

- *Eigenstrain-Based Analysis of Additive Manufacturing Processes*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used to specify the bead dimensions at locations along an arbitrary curve in space (defined using an event series) in a trajectory-based eigenstrain additive manufacturing simulation.

If your trajectory-based eigenstrain additive manufacturing simulation requires time-dependent bead dimensions, you must include a parameter table of type `"ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize"` in the table collection used for element activation. You then use this parameter table to specify the bead height and width at the locations along the tool trajectory.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_EigenStrain_TrajectoryBased_VariableBeadSize", PARAMETERS=2
 INTEGER, 5, "Event Series Field ID for the Bead Height", "Zero Indicates
Constant Bead Height"
 INTEGER, 6, "Event Series Field ID for the Bead Width", "Zero Indicates
Constant Bead Width"
```

Parameters

Event Series Field ID for the Bead Height	Field number in the event series that defines the bead height. Set to 5 by default, indicating that the fifth field in the event series defines the bead height.
Event Series Field ID for the Bead Width	Field number in the event series that defines the bead width. Set to 6 by default, indicating that the sixth field in the event series defines the bead width.

ABQ_AM_MaterialDeposition

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_MaterialDeposition"` in the table collection that you specify to activate elements in the step.

Tables of type `"ABQ_AM_MaterialDeposition"`, `"ABQ_AM_MovingHeatSource"`, and `"ABQ_AM_EigenStrain_TrajectoryBased_Activation"` cannot refer to the same event series in an analysis.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition", PARAMETERS=2
STRING, , "Event Series", ,
STRING, , "Deposition Process", ,"Roller|Bead",
```

Parameters

Event Series	Name of the event series.
Deposition Process	Deposition process type. Options are `"Roller"` or `"Bead"`.

ABQ_AM_MaterialDeposition_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_MaterialDeposition_Advanced"` in the table collection that you specify to activate elements in the step. If a table of this type is not present, default values are used. However, if a table is present, all fields must be explicitly defined. No field should be left blank.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition_Advanced",
PARAMETERS=6
 STRING, "Partial", "Activation Type", , "Full|Partial",
 FLOAT, 0, "Min Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 1, "Max Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 0, "Max Volume Fraction Threshold for Full Activation", , ,
 STRING, "Yes", "Update Orientation", Dimensionless, ,
 INTEGER, "0", "Element Subdivision Order", Dimensionless, ,
```

Parameters

Activation Type

Set to `"Full"` to indicate that elements are activated and their volume fractions set to one. Set to `"Partial"` to indicate that elements are activated when material deposits inside the element, and the volume fractions progressively increase from the minimum volume fraction threshold to one as more material deposits. Default value is `"Partial"`.

Minimum Volume Fraction Threshold for Partial Activation

For activation type `"Partial"`, elements are considered to have at least the minimum threshold volume fraction when any amount of material is deposited inside the element. The minimum volume fraction threshold should be set to a value between zero and one.

Default value is zero.

Maximum Volume Fraction Threshold for Partial Activation

For activation type `"Partial"`, the maximum volume fraction threshold indicates a volume fraction above which element volume fractions are considered to be one.

Default value is one.

Volume Fraction Threshold for Full Activation

For activation type `"Full"`, this volume fraction threshold indicates a volume fraction below which elements are kept inactive and above which elements are activated with a volume fraction of one.

Default value is zero.

Update Orientation

Update orientation for bead-type deposition.

Element Subdivision Order Set to zero for automatic subdivision order.

ABQ_AM_MaterialDeposition_Bead

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_MaterialDeposition_Bead"` in the table collection that you specify to activate elements in the step if the deposition process type is `"Bead"`.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition_Bead", PARAMETERS=5

STRING, , "Stack Direction", , "X|Y|Z",
FLOAT, , "Bead Height", , ,
FLOAT, , "Bead Width", , ,
FLOAT, , "Activation Offset", , ,
STRING, , "Deposition Position", , "Below|Middle|Above",
```

Parameters

Stack Direction	Direction of progressive material deposition layers. Options are `"X"`, `"Y"`, and `"Z"` indicating the global x_g-, y_g-, or z_g-direction.
Bead Height	Height (measured parallel to the stack direction) of the deposited bead of material.
Bead Width	Width (measured perpendicular to the stack direction) of the deposited bead of material.
Activation Offset	Unused. Should be set to zero.
Deposition Position	Set to `"Above"`, `"Middle"`, or `"Below"` to indicate that the deposited material bead is situated above, through, or below the path in the stack direction.

ABQ_AM_MaterialDeposition_Bead_Orientation

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_MaterialDeposition_Bead_Orientation"` in the table collection that you specify to activate elements in the step if the deposition process type is `"Bead"`. You can define an arbitrary fixed bead orientation that is the components of vector ***z***. This orientation vector overrides the stacking direction defined in `"ABQ_AM_MaterialDeposition_Bead"`.

If a table of type `"ABQ_AM_MaterialDeposition_5AxisStrategy"` is present in the same table collection, you must define a time varying orientation vector in the event series defining the toolpath trajectory. The time varying orientation vector overrides the fixed orientation defined in `"ABQ_AM_MaterialDeposition_Bead_Orientation"`.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition_Bead_Orientation",
PARAMETERS=3
 FLOAT, , "X-Component of the Orientation Vector Z", , ,
 FLOAT, , "Y-Component of the Orientation Vector Z", , ,
 FLOAT, , "Z-Component of the Orientation Vector Z", , ,
```

Parameters

X-, Y-, and Z-Components of the Orientation Vector Z	Components are defined with respect to the global coordinate system. Vector ***z*** should not be parallel to the toolpath trajectory.

ABQ_AM_MaterialDeposition_5AxisStrategy

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to define material deposition along an arbitrary curve in space.

You must include a parameter table of type `"ABQ_AM_MaterialDeposition_5AxisStrategy"` in the table collection that you specify to activate elements in the step if the deposition process type is `"BEAD"` and a variable bead orientation is required.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition_5AxisStrategy",
PARAMETERS=4
 STRING, "True", "Ignore Lead Angle/Consider Only Tilt", , "True|False"
 INTEGER, 2, "Event Series Field ID for X-Component of Local Direction
Vector Z1", " "
 INTEGER, 3, "Event Series Field ID for Y-Component of Local Direction
Vector Z1", " "
 INTEGER, 4, "Event Series Field ID for Z-Component of Local Direction
Vector Z1", " "
```

Parameters

Ignore Lead Angle/Consider Only Tilt	If set to `"True"`, Abaqus/Standard considers only the component of the normal vector, $\boldsymbol{n}$, which is perpendicular to the toolpath trajectory. Set to `"True"` if only an approximation of local direction vector, $\boldsymbol{z_1}$, is defined in the event series. Default value is `"True"`.
Event Series Field ID for X-Component of Local Direction Vector Z1	Set to 2 by default, indicating that the second field in the event series defines the x-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Y-Component of Local Direction Vector Z1	Set to 3 by default, indicating that the third field in the event series defines the y-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Z-Component of Local Direction Vector Z1	Set to 4 by default, indicating that the fourth field in the event series defines the z-component of local direction vector $\boldsymbol{z_1}$.

ABQ_AM_MaterialDeposition_VariableBeadSize

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to define the deposition of a bead of material with a varying bead height and width.

You must include a parameter table of type `"ABQ_AM_MaterialDeposition_VariableBeadSize"` in the table collection that you specify for element activation in the step if the deposition process type is `"BEAD"` and a variable bead size is required.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialDeposition_VariableBeadSize",
PARAMETERS=2
 INTEGER, 5, "Event Series Field ID for the Bead Height", "Zero Indicates
Constant Bead Height"
 INTEGER, 6, "Event Series Field ID for the Bead Width", "Zero Indicates
Constant Bead Width"
```

Parameters

Event Series Field ID for the Bead Height	Field number in the event series that defines the bead height. Set to 5 by default, indicating that the fifth field in the event series defines the bead height. Set to zero to indicate that the bead height remains constant with time and equals the nominal height defined in parameter table type *"ABQ_AM_MovingHeatSource_Uniform"* or *"ABQ_AM_MovingHeatSource_Goldak"*.
Event Series Field ID for the Bead Width	Field number in the event series that defines the bead width. Set to 6 by default, indicating that the sixth field in the event series defines the bead width. Set to zero to indicate that the bead width remains constant with time and equals the nominal width defined in parameter table type *"ABQ_AM_MovingHeatSource_Uniform"* or *"ABQ_AM_MovingHeatSource_Goldak"*.

ABQ_AM_MaterialRemoval

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_MaterialRemoval"` in the table collection that you specify to deactivate elements in the step.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval", PARAMETERS=1
STRING, , "Event Series", ,
```

Parameters

Event Series	Name of the event series.

ABQ_AM_MaterialRemoval_5AxisStrategy

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to remove material from a part along an arbitrary curve in space.

You must include a parameter table of type `"ABQ_AM_MaterialRemoval_5AxisStrategy"` in the table collection that you specify to deactivate elements in the step if the deactivation orientation is varying with time.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval_5AxisStrategy",
PARAMETERS=4
 STRING, "False", "Ignore Lead Angle/Consider Only Tilt", , "True|False"
 INTEGER, 2, "Event Series Field ID for X-Component of Local Direction
Vector Z1", " "
 INTEGER, 3, "Event Series Field ID for Y-Component of Local Direction
Vector Z1", " "
 INTEGER, 4, "Event Series Field ID for Z-Component of Local Direction
Vector Z1", " "
```

Parameters

Ignore Lead Angle/Consider Only Tilt	If set to `"True"`, Abaqus/Standard considers only the component of the normal vector, $\boldsymbol{n}$, which is perpendicular to the toolpath trajectory. Set to `"True"` if only an approximation of local direction vector, $\boldsymbol{z_1}$, is defined in the event series. Default value is `"False"`.
Event Series Field ID for X-Component of Local Direction Vector Z1	Set to 2 by default, indicating that the second field in the event series defines the x-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Y-Component of Local Direction Vector Z1	Set to 3 by default, indicating that the third field in the event series defines the y-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Z-Component of Local Direction Vector Z1	Set to 4 by default, indicating that the fourth field in the event series defines the z-component of local direction vector $\boldsymbol{z_1}$.

ABQ_AM_MaterialRemoval_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_MaterialRemoval_Advanced"` in the table collection that you specify to deactivate elements in the step. If a table of this type is not present, default values are used. However, if a table is present, you must explicitly define all fields. No field should be left blank.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval_Advanced", PARAMETERS=5

 STRING, "Partial", "Activation Type", , "Full|Partial",
 FLOAT, 0, "Min Volume Fraction Threshold for Partial Deactivation", , ,
 FLOAT, 1, "Max Volume Fraction Threshold for Partial Deactivation", , ,
 FLOAT, 0, "Max Volume Fraction Threshold for Full Deactivation", , ,
 INTEGER, "0", "Element Subdivision Order", Dimensionless, ,
```

Parameters

Activation Type	Set to `"Full"` to indicate that elements are deactivated and their volume fractions set to one. Set to `"Partial"` to indicate that the volume fractions progressively decrease from the maximum volume fraction threshold to zero as more material is removed. Default value is `"Partial"`.
Minimum Volume Fraction Threshold for Partial Activation	For activation type `"Partial"`, elements are considered to have at least the minimum threshold volume fraction. If any material is removed that decreases the volume fraction below this minimum value, the volume fraction of the element is set to zero and the element is deactivated. You should set the minimum volume fraction threshold to a value between zero and one. Default value is zero.
Maximum Volume Fraction Threshold for Partial Activation	For activation type `"Partial"`, the maximum volume fraction threshold indicates a volume fraction above which element volume fractions are considered to be one. Default value is one.
Volume Fraction Threshold for Full Activation	For activation type `"Full"`, this volume fraction threshold indicates a volume fraction below which elements are deactivated and above which elements are kept active with a volume fraction of one. Default value is zero.
Element Subdivision Order	Set to zero for automatic subdivision order.

ABQ_AM_MaterialRemoval_Bead

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_MaterialRemoval_Bead"` in the table collection that you specify to remove material in the step if the material removal process type is `"Bead"`.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval_Bead", PARAMETERS=5
 STRING, , "Stack Direction", , "X|Y|Z",
 FLOAT, , "Bead Height", , ,
 FLOAT, , "Bead Width", , ,
 FLOAT, , "Unused", , ,
 STRING, ,"Removal Position", , "Below|Middle|Above",
```

Parameters

Stack Direction	Direction of material layers. Options are `"X"`, `"Y"`, and `"Z"` indicating the global x_g-, y_g-, or z_g-direction.
Bead Height	Height, $\boldsymbol{L_3}$, (measured parallel to the stack direction) of the box used for the material removal process.
Bead Width	Width, $\boldsymbol{L_2}$, (measured perpendicular to the stack direction) of the box used for the material removal process.
Activation Offset	Unused. Should be set to zero.
Removal Position	Set to `"Above"`, `"Middle"`, or `"Below"` to indicate that the removed material is situated above, through, or below the path in the stack direction.

ABQ_AM_MaterialRemoval_Bead_Orientation

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_MaterialRemoval_Bead_Orientation"` in the table collection that you specify to deactivate elements in the step. You can define an arbitrary fixed bead orientation that is the components of vector $\boldsymbol{z}$. This orientation vector overrides the stacking direction defined in `"ABQ_AM_MaterialRemoval_Bead"`.

If a table of type `"ABQ_AM_MaterialRemoval_5AxisStrategy"` is present in the same table collection, you must define a time varying orientation vector in the event series defining the toolpath trajectory. The time varying orientation vector overrides the fixed orientation defined in `"ABQ_AM_MaterialRemoval_Bead_Orientation"`.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval_Bead_Orientation",
PARAMETERS=3
 FLOAT, , "X-Component of the Orientation Vector Z", , ,
 FLOAT, , "Y-Component of the Orientation Vector Z", , ,
 FLOAT, , "Z-Component of the Orientation Vector Z", , ,
```

Parameters

X-, Y-, and Z-Components of the Orientation Vector Z	Components are defined with respect to the global coordinate system. Vector $\boldsymbol{z}$ should not be parallel to the toolpath trajectory.

ABQ_AM_MaterialRemoval_VariableBeadSize

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to remove a bead of material of varying height and width from a part.

You must include a parameter table of type `"ABQ_AM_MaterialRemoval_VariableBeadSize"` in the table collection that you specify for element activation in the step if the deposition process type is `"BEAD"` and a variable bead size is required.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MaterialRemoval_VariableBeadSize",
PARAMETERS=2
 INTEGER, 5, "Event Series Field ID for the Bead Height", "Zero Indicates
Constant Bead Height"
 INTEGER, 6, "Event Series Field ID for the Bead Width", "Zero Indicates
Constant Bead Width"
```

Parameters

Event Series Field ID for the Bead Height	Field number in the event series that defines the bead height. Set to 5 by default, indicating that the fifth field in the event series defines the bead height. Set to zero to indicate that the bead height remains constant with time and equals the nominal height defined in parameter table type *"ABQ_AM_MovingHeatSource_Uniform"* or *"ABQ_AM_MovingHeatSource_Goldak"*.
Event Series Field ID for the Bead Width	Field number in the event series that defines the bead width. Set to 6 by default, indicating that the sixth field in the event series defines the bead width. Set to zero to indicate that the bead width remains constant with time and equals the nominal width defined in parameter table type *"ABQ_AM_MovingHeatSource_Uniform"* or *"ABQ_AM_MovingHeatSource_Goldak"*.

ABQ_AM_MovingHeatSource

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type "ABQ_AM_MovingHeatSource" in the table collection for moving heat flux.

Tables of type "ABQ_AM_MaterialDeposition", "ABQ_AM_MovingHeatSource", and "ABQ_AM_EigenStrain_TrajectoryBased_Activation" cannot refer to the same event series in an analysis.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource", PARAMETERS=2
 STRING, , "Event Series", , ,
 STRING, , "Energy Distribution", ,"Concentrated|Uniform|Goldak",
```

Parameters

Event Series	Name of the event series.
Energy Distribution	Set to "Concentrated" to indicate a point energy source, to "Uniform" to indicate that energy is distributed inside a box shape of finite size, or to "Goldak" to indicate that energy is distributed over a three-dimensional space following an exponential expression given by Goldak, et al. (see *Figure 3* in *Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Trajectory-Based Method*).

ABQ_AM_MovingHeatSource_5AxisStrategy

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to define a moving heat source along an arbitrary curve in space.

You must include a parameter table of type `"ABQ_AM_MovingHeatSource_5AxisStrategy"` in the table collection that you specify for a moving heat flux in the step if the deposition process type is `"BEAD"` and a variable bead orientation is required.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_5AxisStrategy",
PARAMETERS=4
 STRING, "True", "Ignore Lead Angle/Consider Only Tilt", , "True|False"
 INTEGER, 2, "Event Series Field ID for X-Component of Local Direction
Vector Z1", " "
 INTEGER, 3, "Event Series Field ID for Y-Component of Local Direction
Vector Z1", " "
 INTEGER, 4, "Event Series Field ID for Z-Component of Local Direction
Vector Z1", " "
```

Parameters

Ignore Lead Angle/Consider Only Tilt	If set to `"True"`, Abaqus/Standard considers only the component of the normal vector, $\boldsymbol{n}$, which is perpendicular to the toolpath trajectory. Set to `"True"` if only an approximation of local direction vector, $\boldsymbol{z_1}$, is defined in the event series. Default value is `"True"`.
Event Series Field ID for X-Component of Local Direction Vector Z1	Set to 2 by default, indicating that the second field in the event series defines the x-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Y-Component of Local Direction Vector Z1	Set to 3 by default, indicating that the third field in the event series defines the y-component of local direction vector $\boldsymbol{z_1}$.
Event Series Field ID for Z-Component of Local Direction Vector Z1	Set to 4 by default, indicating that the fourth field in the event series defines the z-component of local direction vector $\boldsymbol{z_1}$.

ABQ_AM_MovingHeatSource_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type "ABQ_AM_MovingHeatSource_Advanced" in the table collection for moving heat source. If a table of this type is not present, default values are used. However, if a table is present, all fields must be explicitly defined. No field should be left blank.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_Advanced", PARAMETERS=7

 STRING, "False", "Conserve Total Energy", , "True|False",
 STRING, "False" , "Control Increment Size", , "True|False",
 STRING, "Relative", "Offset Type", , "Absolute|Relative",
 FLOAT , 0.0, "VectorX", , ,
 FLOAT , 0.0, "VectorY", , ,
 FLOAT , -1.0, "VectorZ", ,
 FLOAT , 1.0, "Field Factor", , ,
```

Parameters

For reference, see *Figure 1* in *Box Toolpath-Mesh Intersection*.

Conserve Total Energy Set to "True" to indicate that if part of the energy distribution along a path segment is outside the external mesh boundary, the power of the part of the energy distribution that is inside the mesh boundary is proportionally increased to keep the total energy conserved. If set to "False", part of the energy distribution along a path segment outside the external mesh boundary is lost. Default value is "False".

Control Increment Size Unused. Should be set to "False".

Offset Type Set to "Relative" to indicate that the components of the offset vector defined in the table "ABQ_AM_MovingHeatSource_Uniform" are in terms of half of the box lengths in the local x_l-, y_l-, and z_l-directions, respectively. Set to "Absolute" to indicate that the offset values are absolute.

VectorX Global x-component of the local $\boldsymbol{z_1}$ vector for the box shape.

VectorY Global y-component of the local $\boldsymbol{z_1}$ vector for the box shape.

VectorZ Global z-component of the local $\boldsymbol{z_1}$ vector for the box shape.

Field Factor Scaling factor that is multiplied to the first field of the event series. Default value is one. Only used if the energy distribution type is set to "Goldak" or "Uniform".

ABQ_AM_MovingHeatSource_Goldak

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type "ABQ_AM_MovingHeatSource_Goldak" in the table collection for moving heat source if the energy distribution type is set to "Goldak". Parameters defined in this table indicate parameters used in the Goldak (double ellipsoid) expression of laser energy distribution (see *Figure 3* in *Thermomechanical Analysis of Powder Bed–Type Additive Manufacturing Processes Using the Trajectory-Based Method*). It is assumed that material is deposited below the laser path defined by the event series; therefore, you must choose "Below" as the deposition position in the table of type "ABQ_AM_MaterialDeposition_Bead" for a bead-type material deposition method. In this parameter table, you must specify the number of subdivisions (SubDivX, SubDivY, and SubDivZ) used by the subsegment-based approach of the box toolpath-mesh intersection module (see *Box Toolpath-Mesh Intersection*). If you set the number of subdivisions in each direction to 0, Abaqus/Standard automatically computes the number of subdivisions using the dimensional parameters of the Goldak expression and an average element length.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_Goldak", PARAMETERS=10

INTEGER,  , "SubDivX", , ,
INTEGER,  , "SubDivY", , ,
INTEGER,  , "SubDivZ", , ,
FLOAT, , "a", , ,
FLOAT, , "b", , ,
FLOAT, , "cf", , ,
FLOAT, , "cr", , ,
FLOAT, , "ff", , ,
FLOAT, , "fr", , ,
FLOAT, , "BoxSizeFactor", , ,
```

Parameters

For reference, see *Figure 1* in *Box Toolpath-Mesh Intersection*.

SubDivX	Number of subdivisions along the local x_l-direction.
SubDivY	Number of subdivisions along the local y_l-direction.
SubDivZ	Number of subdivisions along the local z_l-direction.
a	Geometric parameter in the Goldak energy distribution expression representing the semi-axis of the ellipsoid in the width direction (that is, along the local y_l-direction.)
b	Geometric parameter in the Goldak energy distribution expression representing the semi-axis of the ellipsoid in the depth direction (that is, along the local z_l-direction).

cf Geometric parameter in the Goldak energy distribution expression representing the semi-axis of the front ellipsoid in the travel direction (that is, along the local x_l-direction).

cr Geometric parameter in the Goldak energy distribution expression representing the semi-axis of the rear ellipsoid in the travel direction (that is, along the local x_l-direction).

ff Front power distribution fraction in the Goldak energy distribution expression.

fr Rear power distribution fraction in the Goldak energy distribution expression.

BoxSizeFactor Scaling factor for the box defined by the geometric parameters in the Goldak energy distribution expression. (Recommended value is 1.0)

ABQ_AM_MovingHeatSource_Goldak_Position

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include a parameter table of type `"ABQ_AM_MovingHeatSource_Goldak_Position"` in the table collection that is used to define the moving heat source. If the energy distribution type is set to `"Goldak"`, you use this parameter table to specify the deposited material bead position along the stack direction.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_Goldak_Position",
PARAMETERS=1
 STRING, "Below", "Deposition Position", , "Below|Middle|Above",
```

Parameters

Deposition Position	Set to `"Above"`, `"Middle"`, or `"Below"` to indicate that the deposited material bead is situated above, through, or below the path in the stack direction.

ABQ_AM_MovingHeatSource_Uniform

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type "ABQ_AM_MovingHeatSource_Uniform" in the table collection for moving heat flux if the energy distribution type is set to "Uniform". Parameters defined in this table indicate lengths, offsets, and subdivisions of the box toolpath used in the toolpath-mesh intersection module.

The parameter table "ABQ_AM_MovingHeatSource_Uniform" includes data that is used to define the geometric characteristics of the applied thermal load. One set of data defines the volume of the box (a rectangular cuboid) over which the applied volumetric flux (units of $JT^{-1}L^{-3}$) is distributed.

You define dimensions (lengths) of the box in the local $\mathbf{x_1}$, $\mathbf{y_1}$, and $\mathbf{z_1}$ directions (BoxLengthX, BoxLengthY, and BoxLengthZ respectively). The box dimensions you specify should be representative of the shape and size of the heat source. You also specify the number of subdivisions in each local direction of the box (SubDivX, SubDivY, and SubDivZ, respectively) used by the subsegment-based approach of the box toolpath-mesh intersection module (see *Box Toolpath-Mesh Intersection*). The full box is divided into a number of smaller boxes based upon the number of subdivisions you specify.

The amount of volumetric flux applied to each smaller box is directly proportional to the ratio of the smaller box volume to the total box volume. For example, if you specify four subdivisions in each local direction, the full box is divided into $(4 \times 4 \times 4) = 64$ smaller boxes and the applied volumetric flux to each of the smaller boxes box is $\frac{1}{64}$ of the total applied volumetric flux.

Heat is applied only to elements that are intersected by the centroid of the smaller boxes. In general, the more subdivisions you define, the more evenly the heat is distributed within your model; however, the computational cost also increases. If you set the number of subdivisions in each direction to 0, Abaqus/Standard automatically computes the number of subdivisions needed to make the volume of each small box equal to the characteristic element volume, which typically results in an acceptably smooth distribution of heat and reasonable computational performance.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_Uniform", PARAMETERS=9

  INTEGER, , "SubDivX", , ,
  INTEGER, , "SubDivY", , ,
  INTEGER, , "SubDivZ", , ,
  FLOAT, , "OffsetX", , ,
  FLOAT, , "OffsetY", , ,
  FLOAT, , "OffsetZ", , ,
  FLOAT, , "BoxLengthX", , ,
  FLOAT, , "BoxLengthY", , ,
  FLOAT, , "BoxLengthZ", , ,
```

Parameters

For reference, see *Figure 1* in *Box Toolpath-Mesh Intersection*.

SubDivX	Number of subdivisions along the local x_l-direction.
SubDivY	Number of subdivisions along the local y_l-direction.
SubDivZ	Number of subdivisions along the local z_l-direction.
OffsetX	Component of the offset vector in the local x_l-direction.
OffsetY	Component of the offset vector in the local y_l-direction.
OffsetZ	Component of the offset vector in the local z_l-direction.
BoxLengthX	Length of the box in the local x_l-direction.
BoxLengthY	Length of the box in the local y_l-direction.
BoxLengthZ	Length of the box in the local z_l-direction.

ABQ_AM_MovingHeatSource_VariableBeadSize

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing to define a moving heat source with a varying height and width.

You must include a parameter table of type `"ABQ_AM_MovingHeatSource_VariableBeadSize"` in the table collection that you specify for a moving heat flux in the step if the deposition process type is `"BEAD"` and a variable bead size is required.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_MovingHeatSource_VariableBeadSize",
PARAMETERS=2
 INTEGER, 5, "Event Series Field ID for the Bead Height", "Zero Indicates
Constant Bead Height"
 INTEGER, 6, "Event Series Field ID for the Bead Width", "Zero Indicates
Constant Bead Width"
```

Parameters

Event Series Field ID for the Bead Height	Set to 5 by default, indicating that the fifth field in the event series defines the bead height.
Event Series Field ID for the Bead Width	Set to 6 by default, indicating that the sixth field in the event series defines the bead width.

ABQ_AM_ThermoMech_Activation_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_ThermoMech_Activation_Advanced"`. Only one set of data can be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_Activation_Advanced",
PARAMETERS=4
 STRING, "Full", "Element Activation", , "Full|Partial",
 FLOAT, 0.0, "Min Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 1.0, "Max Volume Fraction Threshold for Partial Activation", , ,
 FLOAT, 0.0, "Volume Fraction Threshold for Full Activation", , ,
```

Parameters

Element Activation	Set to `"Full"` to indicate that elements are activated and their volume fractions set to one. Set to `"Partial"` to indicate that elements are activated when material deposits inside the element and the volume fractions progressively increase from the minimum volume fraction threshold to one as more material deposits. Default value is `"Full"`.
Minimum Volume Fraction Threshold for Partial Activation	For activation type `"Partial"`, elements are considered to have at least the minimum threshold volume fraction when any amount of material is deposited inside the element. The minimum volume fraction threshold should be set to a value between zero and one. Default value is zero.
Maximum Volume Faction Threshold for Partial Activation	For activation type `"Partial"`, the maximum volume fraction threshold indicates a volume fraction above which element volume fractions are considered to be one. Default value is one.
Volume Fraction Threshold for Full Activation	For activation type `"Full"`, the volume fraction threshold indicates a volume fraction below which elements are kept inactive and above which elements are activated with a volume fraction of one. Default value is zero.

ABQ_AM_ThermoMech_PatternBased_Activation

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_ThermoMech_PatternBased_Activation"` in the table collection. Only one set of data must be defined.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_PatternBased_Activation",
PARAMETERS=13
 FLOAT, 1.0, "Slice Height", , ,
 FLOAT, 1.0, "OriAX", , ,
 FLOAT, 0.0, "OriAY", , ,
 FLOAT, 0.0, "OriAZ", , ,
 FLOAT, 0.0, "OriBX", , ,
 FLOAT, 1.0, "OriBY", , ,
 FLOAT, 0.0, "OriBZ", , ,
 FLOAT, 0.0, "OriCX", , ,
 FLOAT, 0.0, "OriCY", , ,
 FLOAT, 0.0, "OriCZ", , ,
 FLOAT, 0.0, "Recoater Time", , ,
 FLOAT, 0.0, "Print Start Time", , ,
 STRING, "TotalTime", "Total/Step Time", , "TotalTime|StepTime",
```

Parameters

For reference, see *Figure 1*.

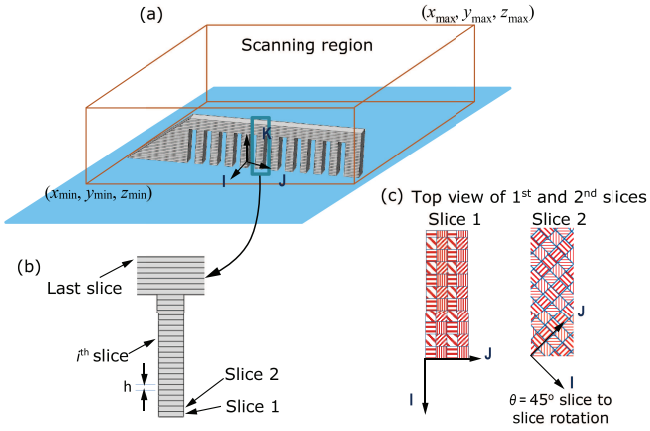


Figure 1: A scan pattern with four pattern patches.

Slice Height	Height of slice for scan pattern.
OriAX, OriAY, OriAZ, OriBX, OriBY, OriBZ, OriCX, OriCY, OriCZ	*X*-, *Y*-, and *Z*-coordinates of points *a*, *b*, and *c* for defining the orientation of the build system *I*–*J*–*K* and its origin. Point *c* is the origin of the system, point *a* must lie on the *I*-axis, and point *b* must lie on the *I*–*J* plane. Although not necessary, it is intuitive to select point *b* such that it is on or near the local *J*-axis.
Recoater Time	Time for the recoater or roller to deposit one layer of raw material.
Print Start Time	Print start time.
Total/Step Time	Set to `"TotalTime"` to indicate that the print start time is in total time. Set to `"StepTime"` to indicate that the print start time is in step time. For more information, see *Time Measures*.

ABQ_AM_ThermoMech_PatternBased_Advanced

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

Optionally, you can include a parameter table of type `"ABQ_AM_ThermoMech_PatternBased_Advanced"` in the table collection. The table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_PatternBased_Advanced",
PARAMETERS=2
 STRING, "SWEEP", "Activation Technique", , "SWEEP|LAYERBYLAYER",
 INTEGER, 1, "Number of Slices per Time Increment", , ,
```

Parameters

Activation Technique	Set to `"LAYERBYLAYER"` to apply heat energy to all elements of a layer/slice simultaneously. Set to `"SWEEP"` to apply heat energy to elements following the scan pattern. This option has an effect only when the increment size is less than the time taken to complete printing one layer.
Number of Slices per Time Increment	Set to the number of layers that will be completed within one increment by automatically adjusting the time increment size. If you set this value to 0 the layer-by-layer time control will be disabled and the standard Abaqus/Standard automatic time control will be used.

ABQ_AM_ThermoMech_PatternBased_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type "ABQ_AM_ThermoMech_PatternBased_Define" in the table collection. Each table must have a unique label. Each table can have multiple sets of data; each set defines one patch/island in a scan pattern.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_PatternBased_Define",
PARAMETERS=6
 FLOAT, 0.0, "xmin", , ,
 FLOAT, 0.0, "ymin", , ,
 FLOAT, 1000.0, "xmax", , ,
 FLOAT, 1000.0, "ymax", , ,
 STRING, " ","Label of an ABQ_AM_ThermoMech_ScanParameter_Define Table", ,
 ,
 FLOAT, 0.0, "Local In-plane Rotation (in degrees)", , ,
```

Parameters

For reference, see *Figure 1*.

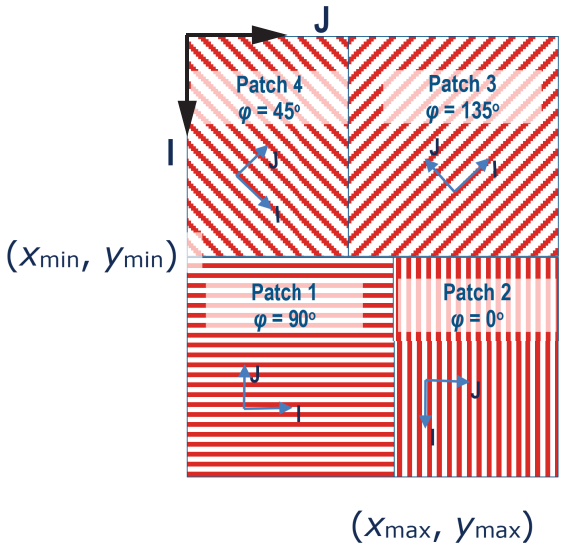


Figure 1: A scan pattern with four patches.

x_{min}, y_{min}, x_{max}, y_{max}	Extent of a scan patch.

Label of an `ABQ_AM_ScanParameter_Define` Table	Refers to the scan parameters that will be applied to this pattern patch.
Local In-plane Rotation	Local in-plane rotation, φ, of the patch (in degrees).

ABQ_AM_ThermoMech_PatternBased_ScanStrategies

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type "ABQ_AM_ThermoMech_PatternBased_ScanStrategies" in the table collection. The table can define multiple sets of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_PatternBased_ScanStrategies",
 PARAMETERS=1
 STRING, " ", "Scan Strategy", , ,
```

Parameters

Scan Strategy Label of a table of type `ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define` or of type `ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset`. Sets of data from this table define the active strategies in the analysis.

ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type "ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define" in the table collection. Each table must have a unique label. Each table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define", PARAMETERS=8
 STRING, " ", "Pattern", , ,
 FLOAT, 0.0, "Relative Rotation Angle (in degrees)", , ,
 FLOAT, -1e36, "Xmin", , ,
 FLOAT, -1e36, "Ymin", , ,
 FLOAT, -1e36, "Zmin", , ,
 FLOAT, 1e36, "Xmax", , ,
 FLOAT, 1e36, "Ymax", , ,
 FLOAT, 1e36, "Zmax", , ,
```

Parameters

For reference, see *Figure 1*.

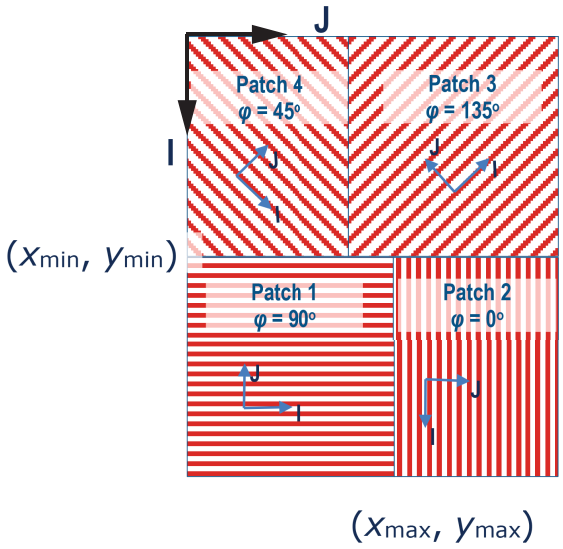


Figure 1: A scan pattern with four patches.

Pattern	Label of a table of type `ABQ_AM_ThermoMech_PatternBased_Define`.
Relative Rotation Angle	Layer-to-layer (slice-to-slice) rotation angle, Θ.
$\mathbf{X_{min}}$, $\mathbf{Y_{min}}$, $\mathbf{Z_{min}}$, $\mathbf{X_{max}}$, $\mathbf{Y_{max}}$, $\mathbf{Z_{max}}$	Extent of a scanning region.

ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You can include multiple parameter tables of type "ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset" in the table collection. Each table must have a unique label. Each table can define only one set of data.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE,
NAME="ABQ_AM_ThermoMech_PatternBased_ScanStrategy_Define_Elset", PARAMETERS=3

 STRING, " ", "Pattern", , ,
 FLOAT,  0.0, "Relative Rotation Angle (in degrees)", , ,
 STRING, " ", "Element Set Name", , ,
```

Parameters

For reference, see *Figure 1*.

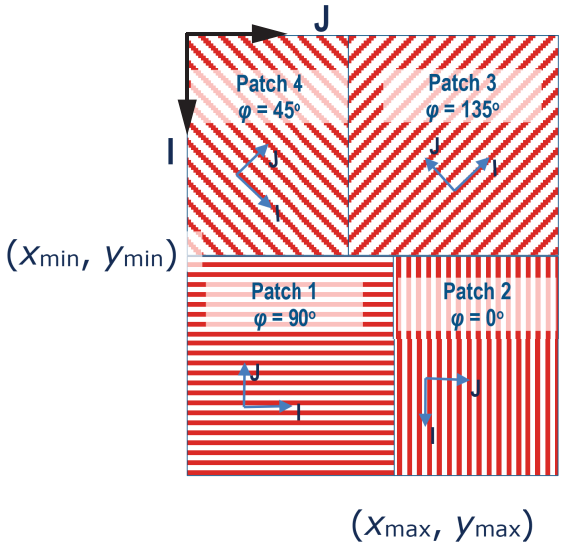


Figure 1: A scan pattern with four patches.

Pattern	Label of a table of type `ABQ_AM_ThermoMech_PatternBased_Define`.
Relative Rotation Angle	Layer-to-layer (slice-to-slice) rotation angle, Θ.

Element Set Name Name of the element set containing the elements qualified to be assigned the scan pattern.

ABQ_AM_ThermoMech_PatternParameters

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_ThermoMech_PatternParameters"` in the table collection.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_PatternParameters",
PARAMETERS=1
 STRING, " ", "Table Collection Name referred to in the *ACTIVATE ELEMENTS
 option", , ,
```

Parameters

Table Collection Name	Name of a table collection referred to in the **ACTIVATE ELEMENTS* option.

ABQ_AM_ThermoMech_ScanParameter_Define

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This parameter table type is used in special-purpose techniques for additive manufacturing.

You must include a parameter table of type `"ABQ_AM_ThermoMech_ScanParameter_Define"` in the table collection. Optionally, you can include multiple tables of this type. Each table must have a unique label. Only one set of data is allowed in each table.

Parameter Table Type Definition

```
*PARAMETER TABLE TYPE, NAME="ABQ_AM_ThermoMech_ScanParameter_Define",
PARAMETERS=4
 FLOAT, 0.0, "Power", , ,
 FLOAT, 0.0, "Hatch Spacing", , ,
 FLOAT, 0.0, "Speed", , ,
 FLOAT, 0.0, "On Time Fraction", , ,
```

Parameters

Power	Power of the laser energy source.
Hatch/Scan Spacing	Spacing between two scan trajectories in a raster pattern.
Speed	Speed at which the laser spot is moving on the layer of raw powder.
On Time Fraction	Fraction of time the laser source is in the "on" state. Set to a value between zero and one.

Event Series and Property Table Type Reference

These event series and property table types are used in special-purpose techniques for additive manufacturing.

In this section:

- *ABQ_AM_AbsorptionCoeff*
- *ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy*
- *ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection*
- *ABQ_AM_HeatSourceTrajectory_RuleID*
- *ABQ_AM_MaterialDeposition*
- *ABQ_AM_MaterialDeposition_5AxisStrategy*
- *ABQ_AM_MaterialDeposition_5AxisStrategy_VariableCrossSection*
- *ABQ_AM_MaterialDeposition_VariableHeightWidth*
- *ABQ_AM_MovingHeatSource_5AxisStrategy_VariableCrossSection*
- *ABQ_AM_PowerMagnitude*
- *ABQ_AM_PowerMagnitude_5AxisStrategy*

ABQ_AM_AbsorptionCoeff

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This property table type is used in special-purpose techniques for additive manufacturing.

You can include a property table of type `"ABQ_AM_AbsorptionCoeff"` in the table collection for the distributed load definition to define the absorption coefficient of the material for laser heating.

Property Table Type Definition

```
*PROPERTY TABLE TYPE, NAME="ABQ_AM_AbsorptionCoeff", PROPERTIES=1
"Absorption Coefficient (between 0 and 1)"
```

Properties

Absorption Coefficient	Absorption coefficient of the material (a value between 0 and 1).

ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy

Products: Abaqus/Standard

References:

- *Eigenstrain-Based Analysis of Additive Manufacturing Processes*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in a trajectory-based eigenstrain additive manufacturing simulation.

An event series of type `"ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy"` defines trajectories of the toolpath for the nozzle and allows for specification of time-dependent tool orientation.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy",
 FIELDS=4
"Eigenstrain Rule Id # (or 0 if tool is off)", DIMENSIONLESS
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
```

Fields

Eigenstrain Rule Id	Set to a nonzero integer value to indicate the id of the eigenstrain rule or to 0 to indicate the "off" state of the tool for the duration of the event series segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.

ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection

Products: Abaqus/Standard

References:

- *Eigenstrain-Based Analysis of Additive Manufacturing Processes*
- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type "ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection" defines trajectories of the toolpath for the nozzle and allows for time-dependent tool orientation and bead dimensions.

Event Series Type Definition

```
*EVENT SERIES TYPE,
NAME="ABQ_AM_EigenStrain_TrajectoryBased_5AxisStrategy_VariableCrossSection",
 FIELDS=6
"Eigenstrain Rule Id # (or 0 if tool is off)", DIMENSIONLESS
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
"Height H", L
"Width W", L
```

Fields

Eigenstrain Rule Id	Set to a nonzero integer value to indicate the id of the eigenstrain rule or to 0 to indicate the "off" state of the tool for the duration of the event series segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.
Height	Defines the time varying bead height.
Width	Defines the time varying bead width.

ABQ_AM_HeatSourceTrajectory_RuleID

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_HeatSourceTrajectory_RuleID"` defines trajectories of the toolpath and eigenstrain rule ID.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_HeatSourceTrajectory_RuleID", FIELDS=1
"Rule ID"
```

Fields

Rule ID	Integer strain rule ID (see parameter table type `ABQ_AM_EigenStrain_TrajectoryBased_Rules`).

ABQ_AM_MaterialDeposition

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_MaterialDeposition"` defines trajectories of the toolpath for material deposition.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_MaterialDeposition", FIELDS=1
"On/Off State"
```

Fields

On/Off State	Set to 1 to indicate the "on" state or to 0 to indicate the "off" state of the tool for the segment.

ABQ_AM_MaterialDeposition_5AxisStrategy

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_MaterialDeposition_5AxisStrategy"` defines trajectories of the toolpath for the nozzle and allows for specification of time varying bead orientation.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_MaterialDeposition_5AxisStrategy", FIELDS=4

"On/Off State", DIMENSIONLESS
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
```

Fields

On/Off State	Set to 1 to indicate the "on" state or to 0 to indicate the "off" state of the tool for the duration of the event series segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.

ABQ_AM_MaterialDeposition_5AxisStrategy_VariableCrossSection

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_MaterialDeposition_5AxisStrategy_VariableCrossSection"` defines trajectories of the toolpath for the nozzle.

Event Series Type Definition

```
*EVENT SERIES TYPE,
NAME="ABQ_AM_MaterialDeposition_5AxisStrategy_VariableCrossSection", FIELDS=6

"On/Off State", DIMENSIONLESS
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
"Height H", L
"Width W", L
```

Fields

On/Off State	Set to 1 to indicate the "on" state or to 0 to indicate the "off" state of the tool for the duration of the segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.
Height	Defines the time varying bead height.
Width	Defines the time varying bead width.

ABQ_AM_MaterialDeposition_VariableHeightWidth

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_MaterialDeposition_VariableHeightWidth"` defines trajectories of the toolpath for the nozzle and allows for specification of time varying bead dimensions.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_MaterialDeposition_VariableHeightWidth",
FIELDS=3
"On/Off State", DIMENSIONLESS
"Height H", L
"Width W", L
```

Fields

On/Off State	Set to 1 to indicate the "on" state or to 0 to indicate the "off" state of the tool for the duration of the event series segment.
Height	Defines the time varying bead height.
Width	Defines the time varying bead width.

ABQ_AM_MovingHeatSource_5AxisStrategy_VariableCrossSection

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_MovingHeatSource_5AxisStrategy_VariableCrossSection"` defines trajectories of the toolpath for the nozzle.

Event Series Type Definition

```
*EVENT SERIES TYPE,
NAME="ABQ_AM_MovingHeatSource_5AxisStrategy_VariableCrossSection", FIELDS=6

"Power Magnitude", ML2T03
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
"Height H", L
"Width W", L
```

Fields

Power Magnitude	Magnitude of the power of the laser for the duration of the event series segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.
Height	Defines the time varying bead height.
Width	Defines the time varying bead width.

ABQ_AM_PowerMagnitude

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_PowerMagnitude"` defines trajectories of the toolpath for the laser.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_PowerMagnitude", FIELDS=1
"Power Magnitude (unit of JT^-1)"
```

Fields

Power Magnitude	Magnitude of the power of the laser for the segment.

ABQ_AM_PowerMagnitude_5AxisStrategy

Products: Abaqus/Standard

References:

- *Special-Purpose Techniques for Additive Manufacturing*

Overview

This event series type is used in special-purpose techniques for additive manufacturing.

An event series of type `"ABQ_AM_PowerMagnitude_5AxisStrategy"` defines trajectories of the toolpath for the nozzle and allows for specification of time varying bead orientation.

Event Series Type Definition

```
*EVENT SERIES TYPE, NAME="ABQ_AM_PowerMagnitude_5AxisStrategy", FIELDS=4
"Power Magnitude", ML2T03
"X-Component of Local Direction Vector Z1", L
"Y-Component of Local Direction Vector Z1", L
"Z-Component of Local Direction Vector Z1", L
```

Fields

Power Magnitude	Magnitude of the power of the laser for the duration of the event series segment.
X-Component of Local Direction Vector Z1	Defines the first component of the local direction vector, z_1.
Y-Component of Local Direction Vector Z1	Defines the second component of the local direction vector, z_1.
Z-Component of Local Direction Vector Z1	Defines the third component of the local direction vector, z_1.

Adaptivity Techniques

In this section:

- *About Adaptivity Techniques*
- *ALE Adaptive Meshing*
- *Adaptive Remeshing*
- *Mesh-to-Mesh Solution Mapping*
- *Fatigue Crack Propagation with Adaptive Remeshing*

About Adaptivity Techniques

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About ALE Adaptive Meshing*
- *About Adaptive Remeshing*
- *Mesh-to-Mesh Solution Mapping*
- *Fatigue Crack Propagation with Adaptive Remeshing*
- **ADAPTIVE REMESH*
- **ADAPTIVE REMESH RETAIN*
- **ADAPTIVE MESH*
- *Understanding adaptive remeshing*

Overview

The finite element discretization that results from suboptimal meshing of models can limit your ability to obtain adequate analysis results at a reasonable CPU cost. This section provides an overview of the adaptivity techniques available in Abaqus that help you optimize a mesh and, therefore, obtain quality solutions while controlling the cost of your analysis. The term "adaptivity" reflects the adaptive, or solution-dependent, processes that Abaqus uses to adapt your mesh to meet your analysis goals.

Selecting an Adaptivity Technique

Four adaptivity techniques are available in Abaqus: Arbitrary Lagrangian-Eulerian (ALE) adaptive meshing; varying topology adaptive remeshing; mesh-to-mesh solution mapping, to enable rezoning analysis; and fatigue crack growth with adaptive remeshing. *Table 1* shows that the adaptivity techniques can be classified according to

- their applicability to achieving particular goals, either accuracy or control of mesh distortion;
- their impact on mesh definitions, either through smoothing a single mesh or through generating multiple dissimilar meshes; and
- when adaptivity occurs with respect to analysis steps.

Table 1: The characteristics of the adaptivity techniques.

	Accuracy	Distortion control	Single mesh	Multiple meshes	Adaptivity occurs
ALE adaptive meshing		√	√		Throughout a step
Adaptive remeshing	√			√	Separately from analysis steps
Mesh-to-mesh solution mapping		√		√	Between analysis steps
Fatigue crack growth with adaptive remeshing	√			√	Separately from analysis steps

ALE Adaptive Meshing

Arbitrary Lagrangian-Eulerian (ALE) adaptive meshing provides control of mesh distortion. ALE adaptive meshing uses a single mesh definition that is gradually smoothed within analysis steps. ALE adaptive meshing is available for limited applications in Abaqus/Standard and is more generally applicable in Abaqus/Explicit. The term ALE implies a broad range of analysis approaches, from purely Lagrangian analysis, in which the node motion corresponds to material motion, to purely Eulerian analysis, in which the nodes remain fixed in space and material "flows" through the elements. Typically ALE analyses use an approach between these two extremes. The ALE feature is distinct from the Eulerian analysis capability in Abaqus/Explicit, which is described in *Eulerian Analysis*.

You can use adaptive meshing to control element distortion in cases where large deformation or loss of material occurs. *Figure 1* illustrates a case where adaptive meshing limits mesh distortion in a bulk forming simulation.

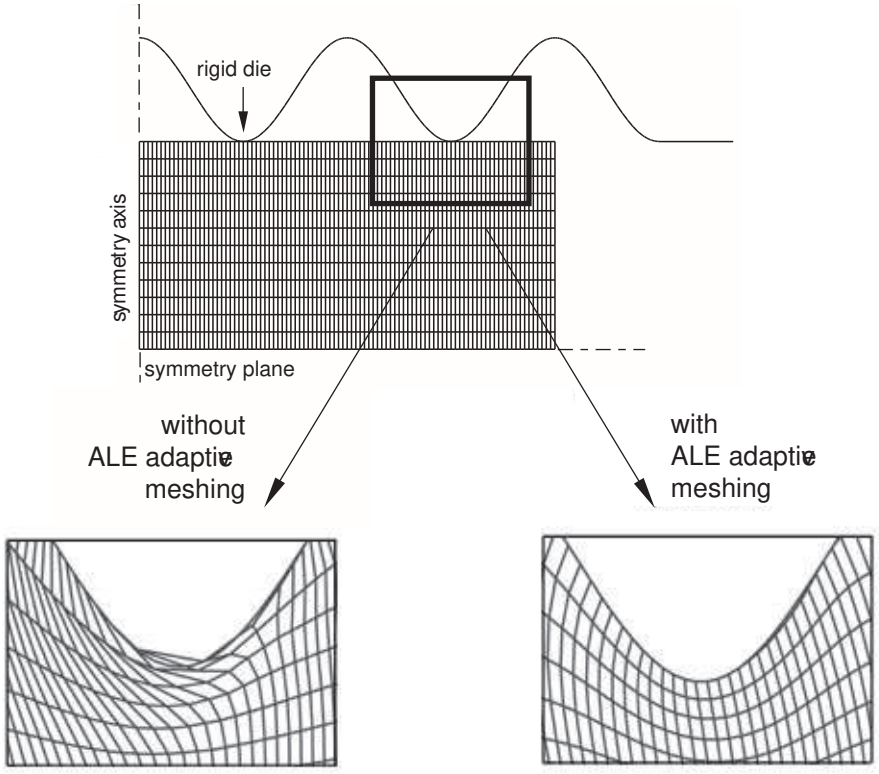


Figure 1: Use of ALE adaptive meshing to control element distortion.

Unlike other adaptivity techniques, adaptive meshing operates on your original mesh definition and is, therefore, useful only when a single mesh can be effective for the duration of a simulation. The mesh is adapted through smoothing of the mesh nodes. This smoothing is typically applied frequently within analysis steps. ALE adaptive meshing requires only one analysis job. See *About ALE Adaptive Meshing* for details.

Adaptive Remeshing (Varying Topology Adaptivity)

Adaptive remeshing is typically used for accuracy control, although it can also be used for distortion control in some situations. The adaptive remeshing process involves the iterative generation of multiple dissimilar meshes to determine a single, optimized mesh that is used throughout an analysis. Adaptive remeshing is available only for Abaqus/Standard analyses submitted from Abaqus/CAE. The goal of adaptive remeshing is to obtain a solution that satisfies mesh discretization error indicator targets that you set, while minimizing the number of elements and, hence, the cost of your analysis. You can use adaptive remeshing to obtain a mesh that provides

a balance between solution cost and desired accuracy. *Figure 2* illustrates a case where adaptive remeshing improves the quality of the stress result around a fillet with targeted mesh refinement.

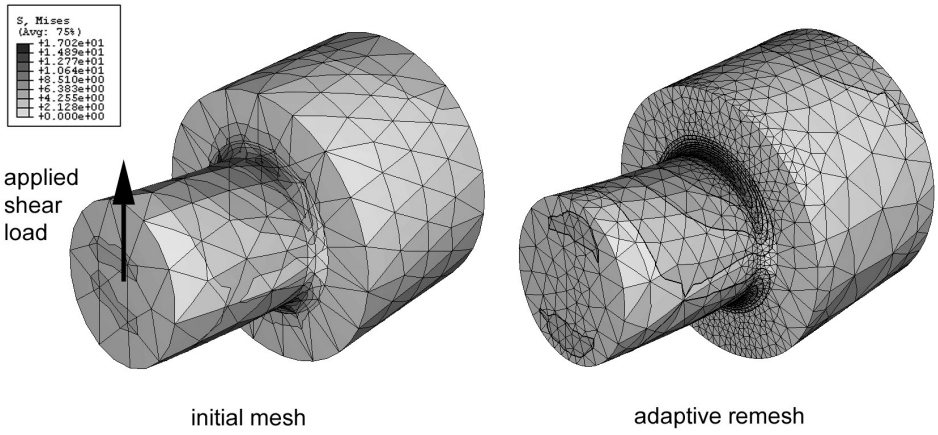


Figure 2: Use of adaptive remeshing to improve the quality of a stress result.

Adaptive remeshing involves an iterative process to determine a single, optimized mesh that is used through an analysis. The iterative process and the remeshing are controlled in Abaqus/CAE. Each successive analysis job covers the same simulation history time period but uses a different mesh. Once the adaptive remeshing process is complete, a single mesh and a single analysis job represent your entire analysis history. See *About Adaptive Remeshing* and *Understanding adaptive remeshing*.

Mesh-to-Mesh Solution Mapping

Mesh-to-mesh solution mapping is available only in Abaqus/Standard. You can use this technique to control element distortion in cases where large deformation occurs by replacing the mesh and continuing the analysis. *Figure 3* illustrates a case where solution mapping is used in conjunction with a new mesh to overcome difficulties associated with element distortion.

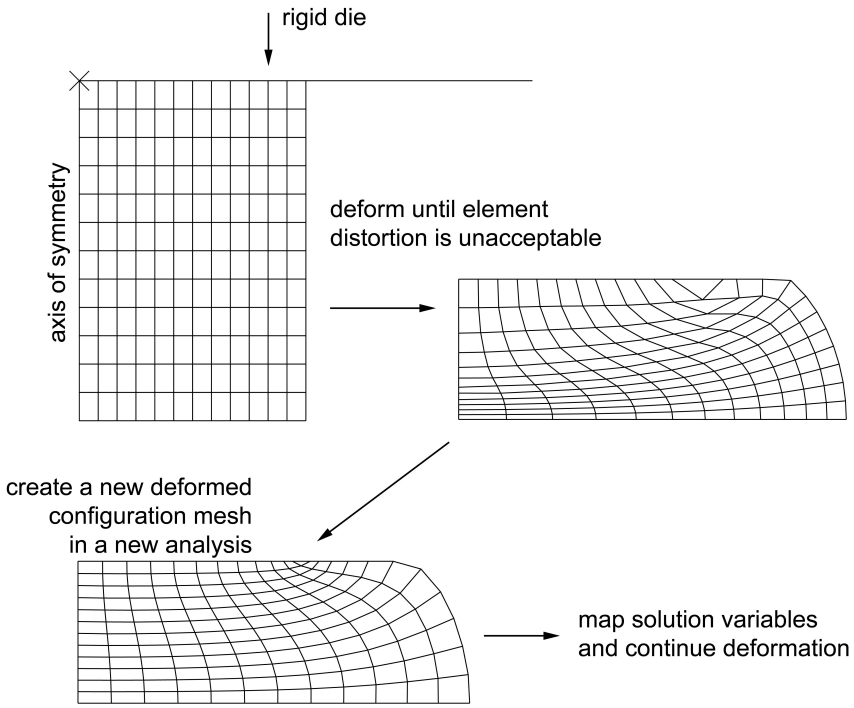


Figure 3: Use of mesh-to-mesh solution mapping as a component of a rezoning technique.

Mesh replacement, or rezoning, involves the creation of multiple Abaqus jobs, each of which represents the configuration of the model in distinct, sequential periods of the simulation history. You use mesh replacement when a single mesh cannot be effective for the duration of a simulation. Each mesh subsequent to the initial configuration reflects a solution-dependent deformed configuration of the model. Therefore, analyses that use mesh replacement are sequentially dependent, and Abaqus uses mesh-to-mesh solution mapping to propagate solution variables from one analysis to the next. In contrast to adaptive remeshing, each mesh replacement job represents a component of the overall analysis history—no single mesh and no single analysis job represent your entire analysis. See *Mesh-to-Mesh Solution Mapping* for details.

Fatigue Crack Growth with Adaptive Remeshing

Fatigue crack growth with adaptive remeshing is based on Abaqus fracture/fatigue technology in conjunction with tetrahedral element–based adaptive remeshing technology. It is available only in Abaqus/Standard.

Abaqus/Standard calculates the stress intensity factors based on the conventional contour integral method (*Contour Integral Evaluation*). The crack growth is governed by either the stress intensity factor or the strain energy release rate–based Paris' law (*Linear Elastic Fatigue Crack Growth Analysis*). The tetrahedral element–based adaptive remeshing technology is used to create a new mesh after each increment of crack growth, such that the new mesh conforms with the updated crack front. The overall fatigue workflow is semi-automated and involves automatic execution (with no user intervention) of a series of successive Abaqus analyses with progressively longer crack lengths. No state mapping is performed or necessary between two successive Abaqus analyses. *Figure 4* illustrates a case where adaptive remeshing is used to create a new mesh that conforms with the updated crack front after crack propagation. For more information on defining fracture onset and growth criteria, controlling the incremental crack growth, and adaptive remeshing, see *Fatigue Crack Propagation with Adaptive Remeshing*.

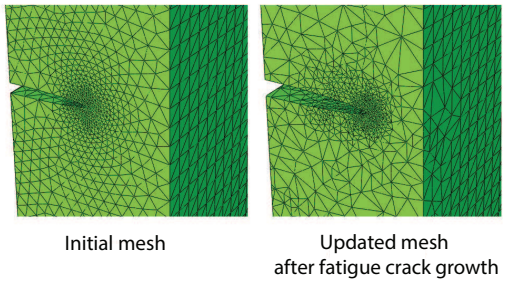


Figure 4: Use of adaptive remeshing to generate a new mesh that conforms with the updated crack front as the crack propagates.

ALE Adaptive Meshing

In this section:

- *About ALE Adaptive Meshing*
- *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*
- *ALE Adaptive Meshing and Remapping in Abaqus/Explicit*
- *Modeling Techniques for Eulerian Adaptive Mesh Domains in Abaqus/Explicit*
- *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*
- *Defining ALE Adaptive Mesh Domains in Abaqus/Standard*
- *ALE Adaptive Meshing and Remapping in Abaqus/Standard*

About ALE Adaptive Meshing

Overview

The adaptive meshing technique in Abaqus combines the features of pure Lagrangian analysis and pure Eulerian analysis.
This type of adaptive meshing is often referred to as Arbitrary Lagrangian-Eulerian (ALE) analysis. The Abaqus documentation often refers to “ALE adaptive meshing” simply as “adaptive meshing.”

ALE adaptive meshing is a tool that makes it possible to maintain a high-quality mesh throughout an analysis, even when large deformation or loss of material occurs, by allowing the mesh to move independently of the material. ALE adaptive meshing does not alter the topology (elements and connectivity) of the mesh, which implies some limitations on the ability of this method to maintain a high-quality mesh upon extreme deformation. Refer to *About Adaptivity Techniques* for a comparison between ALE adaptive meshing and other Abaqus adaptivity methods.

ALE adaptive meshing is distinct from the pure Eulerian analysis capability in Abaqus/Explicit. The pure Eulerian capability supports multiple materials and voids within a single element, which allows effective handling of analyses involving extreme deformation (such as fluid flow). In contrast, ALE elements are always 100% full of a single material; while this formulation limits the deformation of material in the model to the deformation of the elements, it allows more precise definitions of material boundaries and more complex contact interactions. For more information on pure Eulerian analysis, see *Eulerian Analysis*.

Although the adaptive meshing techniques and the user interface are similar in Abaqus/Explicit and Abaqus/Standard, the use-cases and the level of functionality are different. Adaptive meshing in Abaqus/Explicit is intended to model large-deformation problems. It does not attempt to minimize discretization errors in small-deformation analyses. Adaptive meshing in Abaqus/Standard is intended for use in acoustic domains and for modeling the effects of ablation, or wear, of material. A comparison between the adaptive remeshing functionality in Abaqus/Explicit and Abaqus/Standard is provided in this section.

References:

- *ALE Adaptive Meshing*

Features of ALE Adaptive Meshing

ALE adaptive meshing:

- can often maintain a high-quality mesh under severe material deformation by allowing the mesh to move independently of the underlying material; and
- maintains a topologically similar mesh throughout the analysis (that is, elements are not created or destroyed).

In Abaqus/ExplicitALE adaptive meshing:

- can be used to analyze Lagrangian problems (in which no material leaves the mesh) and Eulerian problems (in which material flows through the mesh);
- can be used as a continuous adaptive meshing tool for transient analysis problems undergoing large deformations (such as dynamic impact, penetration, and forging problems);
- can be used as a solution technique to model steady-state processes (such as extrusion or rolling);
- can be used as a tool to analyze the transient phase in a steady-state process;
- can be used in wear analysis to affect underlying element calculations; and

- can be used in explicit dynamics (including adiabatic thermal analysis) and fully coupled thermal-stress procedures.

In Abaqus/StandardALE adaptive meshing:

- can be used to solve Lagrangian problems (in which no material leaves the mesh) and to model effects of ablation, or wear (in which material is eroded at the boundary);
- can be used to update the acoustic mesh when structural preloading causes significant geometric changes in the acoustic domain; and
- can be used in geometrically nonlinear static, steady-state transport, coupled pore fluid flow and stress, and coupled temperature-displacement procedures.

Activating ALE Adaptive Meshing

Adaptive meshing can be applied to an entire model or to individual parts of a model. A Lagrangian adaptive mesh domain will be created, so that the domain as a whole will follow the material originally inside it, which is the proper physical interpretation for most structural analyses. Additional options are provided for controlling the mesh. In Abaqus/Explicit analyses you can define Eulerian boundaries to allow material to flow into or out of the domain modeled.

The subsequent sections of *ALE Adaptive Meshing* describe the various options that can be used with adaptive meshing. Although these options give you the ability to exercise detailed control over adaptive meshing, they are not necessary for many Lagrangian problems.

- To take full advantage of all the adaptive mesh features in Abaqus, it is important to understand the concepts of adaptive mesh domains, boundary regions, boundary edges, geometric features, and mesh constraints. These concepts are explained in *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit* and *Defining ALE Adaptive Mesh Domains in Abaqus/Standard*. Instructions for applying boundary conditions, loads, and surfaces to adaptive mesh boundaries are also provided in those sections.
- *ALE Adaptive Meshing and Remapping in Abaqus/Explicit* and *ALE Adaptive Meshing and Remapping in Abaqus/Standard* outline the methods used to move the mesh and to remap solution variables to the new mesh. These sections also present options for controlling these algorithms. Although the default methods have been chosen to work well for a wide variety of problems, you may wish to override the defaults to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing to relatively difficult or unusual applications.
- Various output and diagnostics are available for verifying the formation of adaptive mesh domains and for interpreting the results of an analysis. These options are explained in *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*.
- *Modeling Techniques for Eulerian Adaptive Mesh Domains in Abaqus/Explicit* gives advice, in the form of examples and modeling hints, on setting up and interpreting Eulerian problems in Abaqus/Explicit using adaptive meshing.

Input File Usage: **ADAPTIVE MESH*, ELSET=*elset_name*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Uses for ALE Adaptive Meshing

Adaptive meshing is of great value in a variety of problems. Abaqus/Explicit and Abaqus/Standard each employ adaptive meshing in ways that provide the greatest value within the particular solver.

Uses in Abaqus/Explicit

In problems where large deformation is anticipated the improved mesh quality resulting from adaptive meshing can prevent the analysis from terminating as a result of severe mesh distortion. In these situations you can use adaptive meshing to obtain faster, more accurate, and more robust solutions than with pure Lagrangian analyses.

Adaptive meshing is particularly effective for simulations of metal forming processes such as forging, extrusion, and rolling because these types of problems usually involve large amounts of nonrecoverable deformation. Because the final shape of the product can be drastically different from the original shape, a mesh that is optimal for the original product geometry can become unsuitable in later stages of the process when large material deformation leads to severe element distortion and entanglement. Element aspect ratios can also degrade in zones with high strain concentrations. Both of these factors can lead to a loss of accuracy, a reduction in the size of the stable time increment, or even termination of the problem.

Uses in Abaqus/Standard

You can use adaptive meshing to enable acoustic domain meshes to follow the large deformations of the bounding structure. In other applications you can use adaptive meshing and adaptive mesh constraints to model arbitrarily large amounts of ablation of material away from the domain.

Adaptive meshing of acoustic regions greatly extends the utility of acoustic analysis procedures. Abaqus can be used to model the response of a coupled structural-acoustic system subjected to structural preloads. By default, the structural-acoustic calculations are based on the original configuration of the acoustic domain. This approximation is adequate as long as the boundary between the fluid and structure does not experience large deformation during application of the preload. However, when the geometry of the acoustic domain changes significantly as a result of structural loading, the original acoustic configuration must be updated. An example is the interior cavity of a tire subjected to inflation, rim mounting, and footprint pressure loads.

The acoustic elements in Abaqus do not have mechanical behavior and, therefore, cannot model the deformation of the fluid when the structure undergoes large deformation. Abaqus/Standard solves the problem of computing the current configuration of the acoustic domain by periodically creating a new acoustic mesh that uses the same topology as the original mesh but with the nodal locations adjusted so that the deformation of the structural-acoustic boundary does not lead to severe distortion of the acoustic elements.

The geometric changes associated with the new acoustic mesh are then taken into account in a subsequent coupled structural-acoustic analysis. However, it is assumed that the material properties of the fluid, such as the density, do not change as a result of mesh smoothing.

Adaptive meshing can also model effects of ablation, or wear, by enabling you to define boundary mesh motions independent of the underlying material motion. An example is the wearing of a tire during its life, an effect that can significantly affect the performance of the structure.

Comparison of ALE Adaptive Meshing in Abaqus/Explicit and Abaqus/Standard

Adaptive meshing in Abaqus/Explicit is generally more robust and provides more features for controlling the mesh than does adaptive meshing in Abaqus/Standard.

ALE Adaptive Meshing in Abaqus/Explicit

Adaptive meshing in Abaqus/Explicit is designed to handle a large variety of problem classes, and employs a variety of smoothing methods, with controls that you can use to tailor the adaptivity to specific problems. The Abaqus/Explicit implementation allows you to do the following:

- to create entirely Eulerian models;
- to improve the quality of the mesh initially, before deformation begins; and
- to define tracer particles, which enable tracking and output of material-based results quantities.

ALE Adaptive Meshing in Abaqus/Standard

Adaptive meshing in Abaqus/Standard uses a single smoothing algorithm that works well for structural acoustic analyses and the modeling of ablation processes. The Abaqus/Standard implementation of adaptive meshing has the following limitations:

- Initial mesh sweeps cannot be used to improve the quality of the initial mesh definition.
- The method is not intended to be used in general classes of large-deformation problems, such as bulk forming.
- Diagnostics capabilities are currently limited.

Illustrative Examples

To illustrate the value of adaptive meshing, simple examples of transient and steady-state forming applications follow. For simplicity, two-dimensional cases are shown. In each case Abaqus/Explicit is used in the simulation.

Axisymmetric Forging

In this example a well-lubricated rigid die of sinusoidal shape moves down to deform a blank of rectangular cross-section (see *Figure 1*).

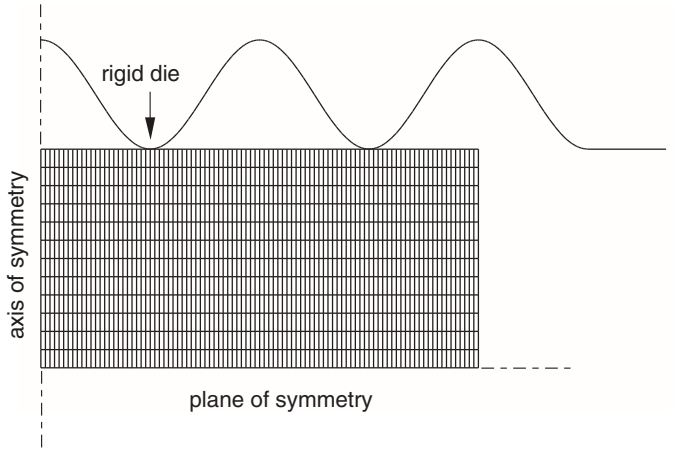


Figure 1: A blank and a sinusoidal die.

The indentation depth is 80% of the original blank thickness. Material extrudes upward and outward (radially) as the blank is indented. The die is modeled with an analytical rigid surface, and the blank is modeled with axisymmetric continuum elements in a regular mesh configuration. The blank is assumed to have elastic-plastic material properties.

A pure Lagrangian analysis of this problem does not run to completion because of excessive distortion in several elements, as shown in *Figure 2*. The contact surface cannot be treated correctly because of the gross distortion of the elements at the troughs of the sinusoidal rigid surface.

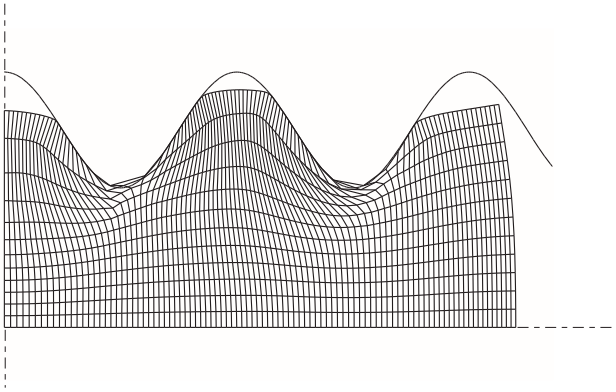


Figure 2: Eventually, the purely Lagrangian analysis will terminate because of excessive element distortion.

Adaptive meshing allows the problem to run to completion. A Lagrangian adaptive mesh domain is created for the entire blank. Abaqus/Explicit automatically chooses suitable defaults for adaptive meshing; hence, the adaptive mesh approach requires only two additional input lines:

```
*HEADING
 ...
*ELSET, ELSET=BLANK
**************************
*STEP
*DYNAMIC, EXPLICIT
 ...
*ADAPTIVE MESH, ELSET=BLANK
 ...
*END STEP
```

Figure 3 and *Figure 4* show the deformed mesh at various stages of the forming analysis. Because the mesh refinement is maintained on the areas of the secondary surface that contact the die troughs as the material flows radially, contact conditions are resolved correctly throughout the analysis.

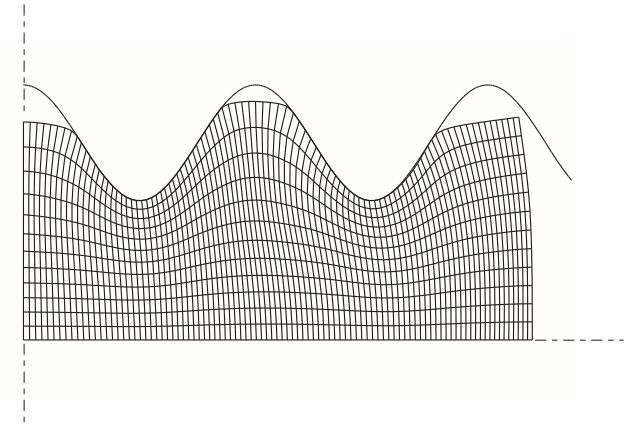


Figure 3: Deformed configuration at an intermediate stage of the analysis.

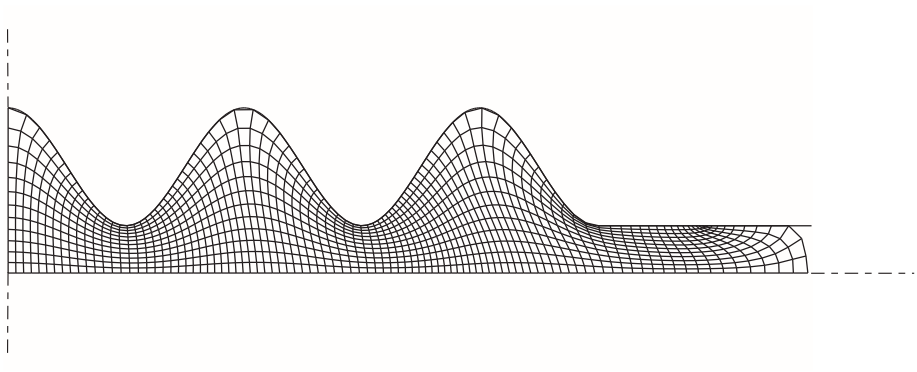


Figure 4: Deformed configuration upon completion of the analysis.

Steady-State Rolling Example

This example shows how adaptive meshing can be used in a steady-state simulation to allow the flow of material through Eulerian boundaries on the problem domain. A steel plate is passed through a symmetric roll stand to reduce its height by 50%. This simulation is run until it reaches steady-state conditions.

Figure 5 and *Figure 6* show the initial and final (steady-state) configurations in a purely Lagrangian model of this problem.

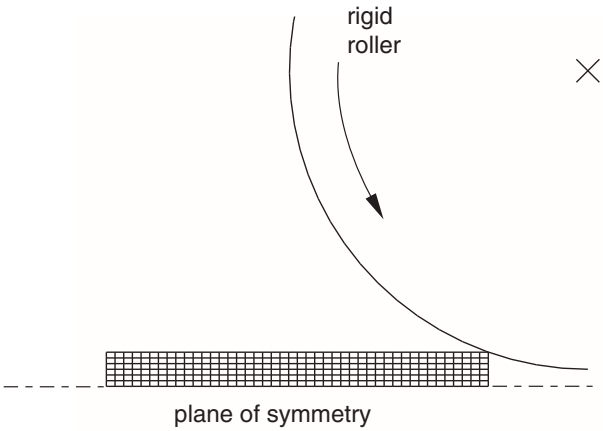


Figure 5: The initial configuration of the roller and the undeformed blank in the pure Lagrangian model.

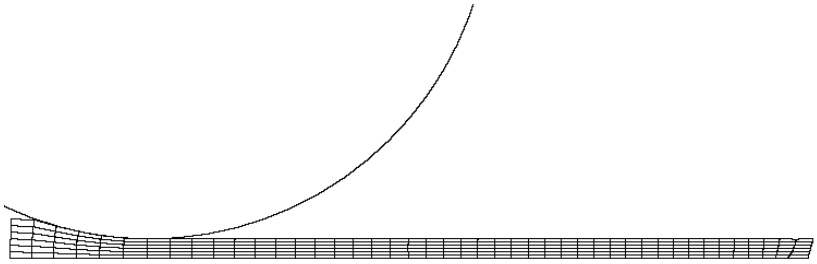


Figure 6: The final steady-state configuration in the pure Lagrangian model.

Figure 7 shows this problem modeled using an Eulerian adaptive mesh domain, where material flows through the mesh.

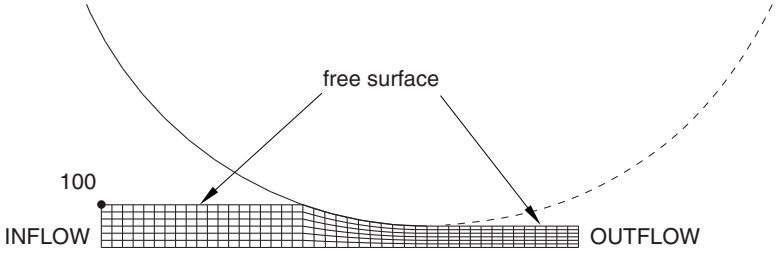


Figure 7: The initial Eulerian adaptive mesh domain.

Only the region near the roller is modeled. The exact location of the free surface does not need to be known to set up the problem: it is created in a likely location, and the final steady-state position is found as part of the solution. Although not shown, a focused mesh can be used to capture steep strain gradients directly beneath the roller. The Eulerian domain reaches the same steady-state solution as obtained with the Lagrangian approach.

The Eulerian adaptive mesh domain is created by defining an inflow and an outflow boundary on the adaptive mesh domain. Adaptive mesh constraints are applied normal to these boundaries so that material will flow through the mesh (see *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*). Frictional contact between the roller and the blank pulls material through the adaptive mesh domain.

The problem is set up by making the following modifications to the input file for the pure Lagrangian analysis:

```
*HEADING
 ...
*ELSET, ELSET=BILLET
 ...
*ELSET, ELSET=INFLOW
 ...
*ELSET, ELSET=OUTFLOW
 ...
*NSET, NSET=INFLOW
 ...
*NSET, NSET=OUTFLOW
 ...
*SURFACE, NAME=INFLOW, REGION TYPE=EULERIAN
 INFLOW, S1
*SURFACE, NAME=OUTFLOW, REGION TYPE=EULERIAN
 OUTFLOW, S2
**************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
 ...
*ADAPTIVE MESH, ELSET=BILLET, CONTROLS=ADAPT
*ADAPTIVE MESH CONTROLS, NAME=ADAPT
*ADAPTIVE MESH CONSTRAINT, TYPE=DISPLACEMENT
 INFLOW, 1, 1, 0.0
 100, 2, 2, 0.0
 OUTFLOW, 1, 1, 0.0
 ...
*END STEP
```

Adaptive mesh controls were not required to solve this problem; they were included for illustrative purposes (see *ALE Adaptive Meshing and Remapping in Abaqus/Explicit* for details).

Defining ALE Adaptive Mesh Domains in Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References:

- *About ALE Adaptive Meshing*
- *ALE Adaptive Meshing and Remapping in Abaqus/Explicit*
- **ADAPTIVE MESH*
- *Understanding ALE adaptive meshing*

Overview

Arbitrary Lagrangian-Eulerian (ALE) adaptive mesh domains:

- define the portions of a finite element model where mesh movement is independent of material deformation;
- can be used to analyze Lagrangian or Eulerian problems;
- can contain only first-order, reduced-integration, solid elements (4-node quadrilaterals, 3-node triangles, 8-node hexahedra, 6-node wedges, and 4-node tetrahedra);
- can be used in planar, axisymmetric, and three-dimensional geometries;
- have boundary regions where loads, boundary conditions, and surfaces can be defined; and
- are active only for geometrically nonlinear steps.

Defining an ALE Adaptive Mesh Domain

ALE adaptive meshing is performed in adaptive mesh domains, which can be either Lagrangian or Eulerian. Within either type of adaptive mesh domain the mesh will move independently of the material. Lagrangian adaptive mesh domains are usually used to analyze transient problems with large deformations. On the boundary of a Lagrangian domain the mesh will follow the material in the direction normal to the boundary, so that the mesh covers the same material domain at all times. Eulerian adaptive mesh domains are usually used to analyze steady-state processes involving material flow. On certain user-defined boundaries of an Eulerian domain, material can flow into or out of the mesh. By default, the mesh is not fixed spatially on these boundaries; mesh constraints must be applied to prevent the mesh from moving with the material, as described in *Mesh Constraints*, presented later in this section. There can never be any “empty” elements; all elements in the domain must be filled completely with material at all times.

You must specify the region of the original mesh that will be subject to adaptive meshing.

Input File Usage: **ADAPTIVE MESH*, ELSET=*name*

Multiple adaptive mesh domains can be defined in a step by reusing the **ADAPTIVE MESH* option (for example, to prevent material from flowing from one domain to another or to apply adaptive meshing to unconnected domains). The element sets used to create adaptive mesh domains cannot overlap.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Only one adaptive mesh domain can be defined in Abaqus/CAE for any particular step.

Modifying an ALE Adaptive Mesh Domain

By default, all adaptive mesh domains defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh domains in effect for a given step relative to the preexisting adaptive mesh domains. At each new step the existing adaptive mesh domains can be modified and additional adaptive mesh domains can be specified (except in Abaqus/CAE, where only one adaptive mesh domain can be in effect for a given step).

Input File Usage: Use either of the following options to modify an existing adaptive mesh domain or to specify an additional adaptive mesh domain:

**ADAPTIVE MESH*, ELSET=*name*
**ADAPTIVE MESH*, ELSET=*name*, OP=MOD

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**

Removing an ALE Adaptive Mesh Domain

If you choose to remove any adaptive mesh domain in a step, no adaptive mesh domains will be propagated from the previous step. Therefore, all adaptive mesh domains that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh domains and to specify new adaptive mesh domains:

**ADAPTIVE MESH*, ELSET=*name*, OP=NEW

If the OP=NEW parameter is used on any **ADAPTIVE MESH* option within a step, it must be used on all **ADAPTIVE MESH* options in the step.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **No adaptive mesh domain for this step**

Splitting ALE Adaptive Mesh Domains

User-defined adaptive mesh domains are examined by Abaqus/Explicit. The user-defined domain will be modeled using a single adaptive mesh if the domain:

- consists of a single element type;
- consists of a single connected region;
- consists of a single material;
- is subject to a uniform body force (including zero body force); and
- has identical section controls.

The user-defined domain will be split into multiple adaptive mesh domains, separated by boundary regions, if the domain:

- consists of multiple element types;
- spans part instances;

- consists of multiple regions (including regions that are connected by less than a single element face, only by contact conditions, or only by connectors such as MPCs);
- consists of multiple materials;
- is subject to multiple body force definitions; or
- is subject to multiple section control definitions.

In this documentation the term "adaptive mesh domain" refers to a single domain after splitting by Abaqus/Explicit. On the rare occasion that a reference is made to an adaptive mesh domain prior to the automatic splitting, it is referred to as a "user-defined adaptive mesh domain." Since adaptive mesh domains are split across element types, degenerate elements should be used for mixed domains that include both triangles and quadrilaterals (or tetrahedron and bricks). For example, when defining a mixed plane strain domain with quadrilateral and triangular elements, the CPE4R element type should be used to define both quadrilaterals and degenerated quadrilaterals. Using the CPE3 element will result in split domains, which is generally not desirable.

ALE Adaptive Mesh Boundary Regions

Each ALE adaptive mesh domain has a boundary, which can consist of one or more regions. (Regions, in this context, are surfaces in three-dimensional models or lines in two-dimensional or axisymmetric models.) A boundary region can be either Lagrangian, sliding, or Eulerian. Some boundary regions are created automatically by Abaqus/Explicit; others can be created by defining boundary conditions, loads, and surfaces. Adaptive mesh boundary regions are separated by edges in three dimensions and by corners in two dimensions. Both edges and corners are referred to as "boundary region edges" throughout this documentation.

Boundary Region Edges

Two types of boundary region edges can exist: Lagrangian and sliding. Lagrangian edges are always associated with a material line. Material can never flow past a Lagrangian edge, and nodes can move only along a Lagrangian edge (like beads on a string). Sliding edges are associated only with the mesh. Material can flow past a sliding edge (that is, sliding edges are free to slide over the underlying material).

Lagrangian edges can be viewed with Abaqus/CAE; see *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*.

Lagrangian Boundary Regions

Lagrangian boundary regions are the most common boundary regions in structural finite element analysis; therefore, with the exception of contact surfaces, they are always the default in Abaqus/Explicit. A Lagrangian boundary region has the most constraints of all the boundary region types. The mesh is constrained to move with the material in the direction normal to the surface of the boundary region and in the directions perpendicular to the boundary region edges.

Lagrangian boundary regions have Lagrangian edges: the edges follow the material. On the interior of a Lagrangian boundary region, the mesh can move independently of the material in the surface of the boundary region. Thus, a Lagrangian boundary region can be thought of as a "mesh patch" that follows the material. Nodes are free to move within and along the edges of the patch but cannot leave the patch.

Lagrangian Corners

A Lagrangian corner is formed where two Lagrangian edges meet. The node at a Lagrangian corner is constrained to move with the material in all directions; it is nonadaptive.

Sliding Boundary Regions

A sliding boundary region is the same as a Lagrangian boundary region except that it has a sliding edge. Sliding boundary regions are created by default when you define a surface on the boundary of an adaptive mesh domain (see *About Surfaces*).

The mesh is constrained to move with the material in the direction normal to the boundary region, but it is completely unconstrained in the directions tangential to the boundary region. Thus, a sliding boundary region can be thought of as a "mesh patch" that moves independently of the underlying material.

Sliding boundary regions can be created by defining a surface, boundary condition, or load on the boundary of an adaptive mesh domain (as explained later in this section). Since the mesh is totally unconstrained in the directions tangential to a sliding boundary region, the location of an applied boundary condition or load may not be physically meaningful as the mesh moves over the material. Therefore, to retain the spatial meaning of an applied boundary condition or load, spatial mesh constraints (described in *Mesh Constraints*, presented later in this section) are usually applied tangential to sliding boundary regions.

Eulerian Boundary Regions

Eulerian boundary regions can be defined on the exterior of a model where it makes physical sense to let material flow across the boundary (for example, at the inlet and outlet of a steady-state extrusion or rolling problem). This flow across the boundary distinguishes Eulerian boundary regions from Lagrangian or sliding boundary regions.

Eulerian boundary regions have sliding edges and must lie completely on an exterior surface of a model. It makes no physical sense to allow material flow to originate on an interior surface. You must explicitly define Eulerian boundary regions since, by default, Abaqus/Explicit assumes that no material flows into or out of an adaptive mesh domain.

Eulerian boundary regions are created by defining a surface, a boundary condition, or a load on the boundary of an adaptive mesh domain. On Eulerian boundary regions the mesh motion usually should be constrained in the direction normal to the material motion; therefore, the surface mesh should be fixed in space using spatial mesh constraints (described in *Mesh Constraints*, presented later in this section). Applying these constraints normal to an Eulerian boundary region allows material to flow into or out of the mesh, as in a fluid flow problem, while allowing adaptive meshing to occur on the surface of the boundary region to maximize mesh quality.

The material flowing into an Eulerian boundary region is assumed to have the same properties as the material that is inside the adaptive mesh domain.

Techniques for modeling Eulerian domains are presented in *Modeling Techniques for Eulerian Adaptive Mesh Domains in Abaqus/Explicit*.

Creation of Boundary Regions

Abaqus/Explicit will create adaptive mesh boundary regions automatically on

- the exterior of a model,
- the boundary between different adaptive mesh domains, or
- the boundary between an adaptive mesh domain and a nonadaptive domain.

By default, a boundary region on the exterior of a model will be Lagrangian, so that the boundary region follows the material, and loads, boundary conditions, etc. will retain their Lagrangian interpretation. A boundary region between different adaptive mesh domains is always Lagrangian: no material can flow through such a boundary region. A boundary region between an adaptive mesh domain and a nonadaptive domain is always nonadaptive. The only exception to this occurs if an Eulerian boundary region is defined on the boundary between an adaptive mesh domain and a nonadaptive domain that comprises displacement-based infinite elements. In this case the

nodes on the boundary behave as in Eulerian boundary regions (see the description under *Eulerian Boundary Regions*, presented earlier in this section), and the mesh motion at the boundary nodes can be constrained using spatial mesh constraints.

The boundary between two different materials can never "flow" through the mesh; such a physical boundary is always associated with a Lagrangian boundary region or a nonadaptive mesh boundary.

Figure 1 shows some boundary regions that will be created automatically by Abaqus/Explicit. In the model shown in this figure Abaqus/Explicit splits the user-defined adaptive mesh domain into two adaptive mesh domains because the original domain is composed of two different materials.

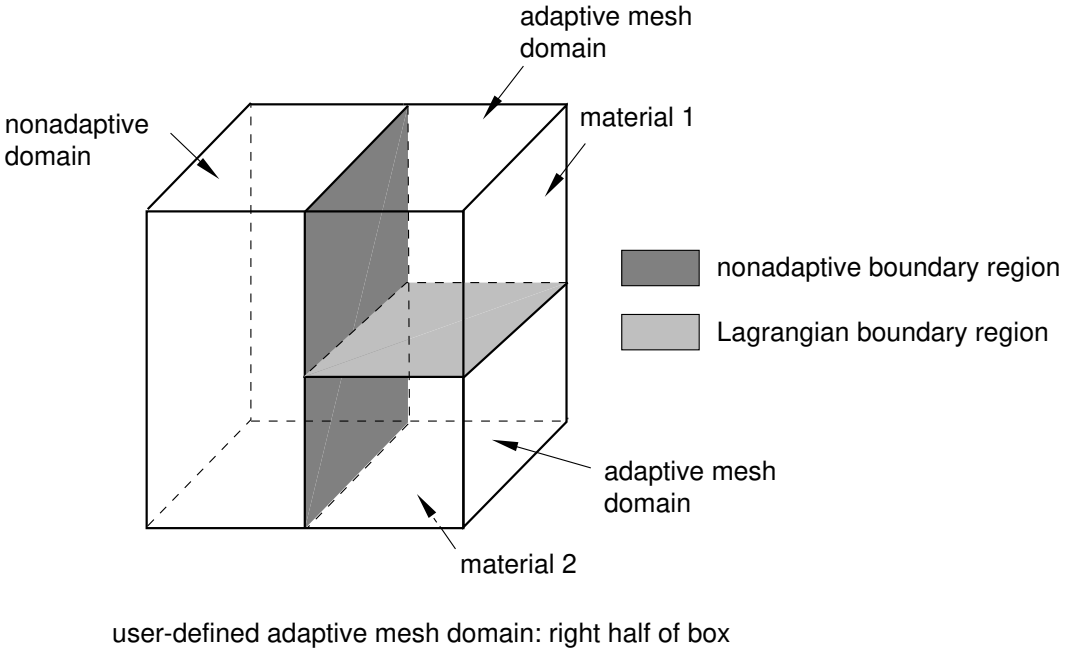


Figure 1: Automatic splitting of mesh domains and creation of boundary regions.

In addition to the boundary regions created automatically by Abaqus/Explicit, Lagrangian, sliding, and Eulerian boundary regions can be created by the definition of surfaces, boundary conditions, and loads, as described later in this section.

Geometric Features

Many models include distinct geometric kinks that take the form of geometric edges or corners. It is usually not desirable to perform adaptive meshing across such geometric features unless they flatten. Once a geometric feature does flatten, it is usually best if the feature is deactivated so that adaptive meshing will occur across it. This is especially true when adaptive mesh domains are subject to large deformation.

The adaptive meshing algorithm in Abaqus/Explicit will respect geometric features on Lagrangian and sliding boundaries. In three dimensions geometric features consist of edges and corners (see *Figure 2*), while in two dimensions they consist of only corners. If a geometric edge coincides with the edge of a Lagrangian boundary region, the presence of the geometric feature has no effect on the treatment of the edge: material cannot flow perpendicular to a Lagrangian edge.

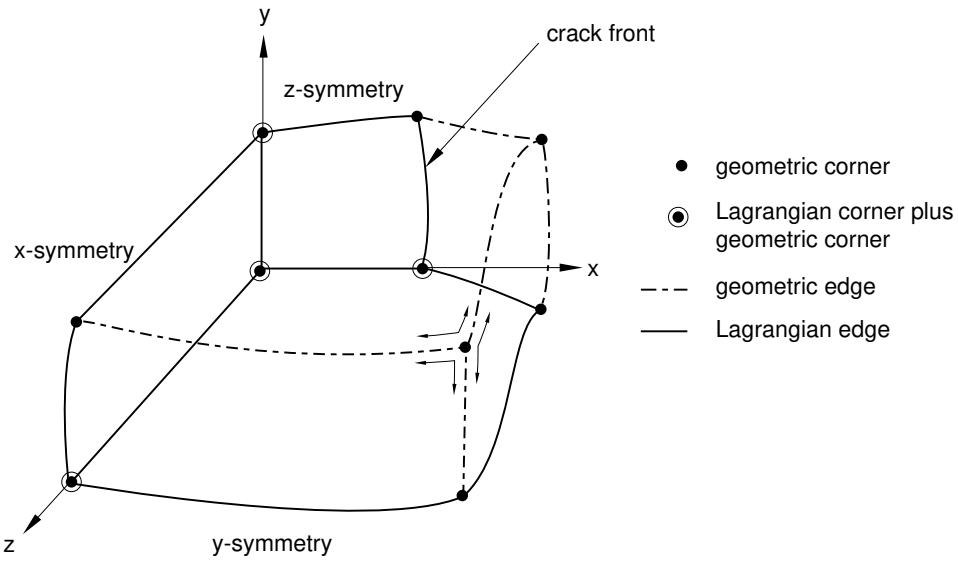


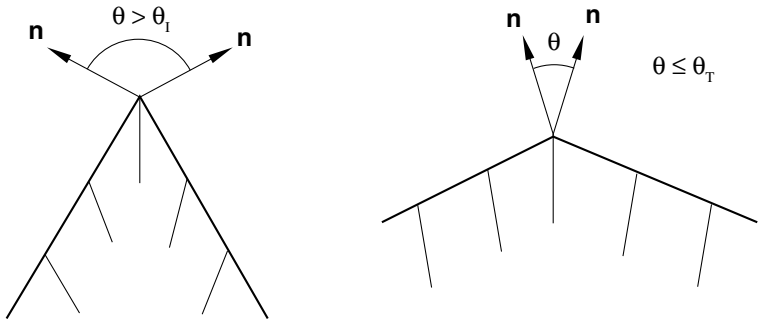
Figure 2: Geometric features formed on a solid block with a crack.

Geometric features are not detected or tracked on Eulerian boundary regions because they generally are not physically meaningful.

Output options are available for viewing the formation of geometric edges and corners—see *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*.

Controlling the Detection of Geometric Edges and Corners

Geometric features are identified initially as edges on boundary regions where the angle between the normals on adjacent element faces is greater than the initial geometric feature angle, θ_I ($0° \le \theta_I \le 180°$). See *Figure 3*. The default value for the initial geometric feature angle is $\theta_I = 30°$.



Initial mesh with a geometric feature: no mesh flow is permitted past the corner.

The geometric feature is deactivated during simulation.

Figure 3: Detection and deactivation of geometric features.

You can change the value of the angle that will be used to recognize geometric features. Setting $\theta_I = 180°$ will ensure that no geometric edges or corners are formed on the boundary of the adaptive mesh domain.

Input File Usage: **ADAPTIVE MESH CONTROLS*, NAME=*name*,
INITIAL FEATURE ANGLE=θ_I

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Initial feature angle:** θ_I

Controlling the Deactivation of Geometric Edges and Corners

Geometric features affect only Lagrangian and sliding boundary regions, where they act as temporary Lagrangian edges. During each mesh sweep in an adaptive mesh increment, nodes along a geometric edge are positioned by applying the basic smoothing methods (see *ALE Adaptive Meshing and Remapping in Abaqus/Explicit*). The nodes are constrained to lie along the discrete geometric edge unless the angle forming the geometric edge becomes less than the transition geometric feature angle, θ_T ($0° \le \theta_T \le 180°$). The default value for the transition feature angle is $\theta_T = 30°$. If the angle across the geometric edge becomes less than θ_T, the boundary surface is considered to be flattened sufficiently for the feature to be deactivated, and the mesh is allowed to slide freely over the material (unconstrained by the deactivated geometric edge). Geometric corners are allowed to flatten in a similar fashion. This logic allows great flexibility in mesh adaptation while preserving geometric features in the model.

You can change the transition feature angle. Setting $\theta_T = 0°$ will ensure that no geometric edges or corners are ever deactivated.

Input File Usage: **ADAPTIVE MESH CONTROLS*, NAME=*name*,
TRANSITION FEATURE ANGLE=θ_T

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Transition feature angle:** θ_T

Mesh Constraints

In most adaptive mesh problems the motion of nodes in the mesh is determined by the meshing algorithm, with constraints imposed by the domain boundary and the boundary region edges. However, there are cases when you must explicitly define the motion of the nodes. As explained earlier, Eulerian and sliding boundary regions generally require mesh constraints for the regions to be physically meaningful. In some problems you may wish to keep certain nodes fixed, to move nodes in a particular direction, or to force certain nodes to move with the material. In other problems you may desire a node or particular set of nodes to follow the material motion. Adaptive mesh constraints allow full control over the mesh movement and act independently of any boundary conditions or loads applied to the underlying material.

Applying Spatial Mesh Constraints

Use a spatial mesh constraint (the default) to prescribe spatial mesh motion that is independent of the material motion. You specify the nodes to which the constraint is applied, the directions of the prescribed motion, and the amplitude of the prescribed motion. You can prescribe either a displacement or a velocity for the spatial mesh motion.

Input File Usage: Use the following option to define the mesh constraints explicitly:

**ADAPTIVE MESH CONSTRAINT*, CONSTRAINT TYPE=SPATIAL,
TYPE=DISPLACEMENT or VELOCITY

Abaqus/CAE Usage: To define the mesh constraints explicitly:

Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**

Rules for Applying Spatial Mesh Constraints

Spatial mesh constraints can be applied without restriction to nodes on an Eulerian boundary region or in the interior of an adaptive mesh domain.

In both two and three dimensions nodes at Lagrangian and active geometric corners are fully constrained to move with the underlying material. No mesh constraints can be applied at such corners.

Adaptive mesh constraints must not conflict with other mesh constraints inherent to Lagrangian and sliding boundary regions; therefore, adaptive mesh constraints can be applied only tangentially to a Lagrangian or sliding boundary region. This restriction implies that adaptive mesh constraints can be applied only in two directions in a three-dimensional boundary region, in one direction in a two-dimensional boundary region, or in one direction on a Lagrangian or active geometric edge. It may not always be feasible to adhere to this rule, particularly if the boundary experiences finite rotation. Therefore, if the normal to a boundary region is not perpendicular to a prescribed mesh constraint at a node, it is always moved along the current surface of the boundary region so that the projection of the mesh motion in the direction of the prescribed constraint is correct (see *Figure 4*).

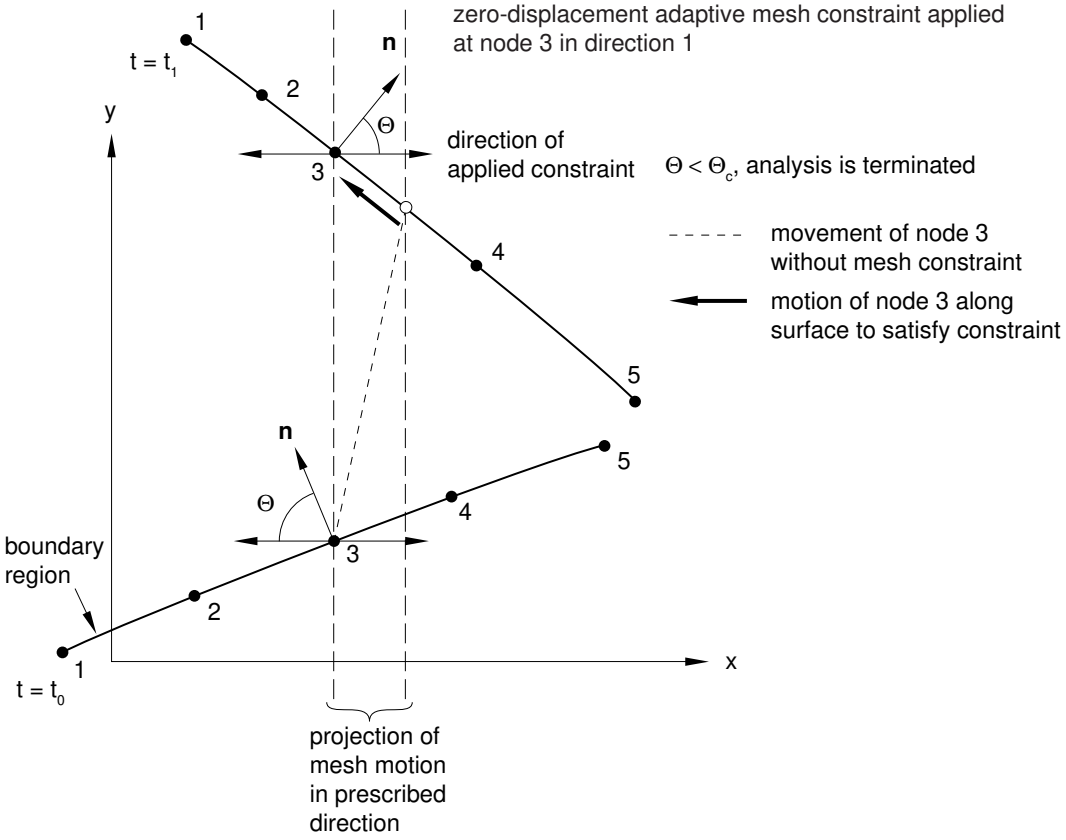


Figure 4: Enforcing a spatial mesh constraint.

If the normal to the boundary region approaches the direction of the applied mesh constraint, large mesh motions will be required to satisfy the constraint. By default, an analysis is terminated if the angle between the normal to the boundary region and the direction of the prescribed constraint becomes less than θ_C. This cutoff angle is referred to as the mesh constraint angle, and its default value is 60°.

The mesh constraint angle, θ_C, is also used when adaptive mesh constraints are applied to nodes along a Lagrangian or active geometric edge. Since independent mesh motion cannot be prescribed perpendicular to these edges, an analysis is terminated if the angle between the prescribed constraint and the plane perpendicular to the edge falls below the specified mesh constraint angle.

You can change the value of the mesh constraint angle ($5° \le \theta_C \le 85°$). Setting $\theta_C < 45°$ is not recommended because it may cause errors in determining the free surface geometry, especially for curved surfaces.

Input File Usage: **ADAPTIVE MESH CONTROLS*, MESH CONSTRAINT ANGLE=θ_C

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Mesh constraint angle:** θ_C

Defining Mesh Constraints That Vary with Time

The prescribed magnitude of a nonzero mesh constraint can vary with time during a step according to an amplitude definition (see *Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**ADAPTIVE MESH CONSTRAINT*, AMPLITUDE=*name*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**: **Amplitude:** *amplitude*

Applying Spatial Mesh Constraints in Local Directions

Spatial mesh constraints are applied in local directions if a local coordinate system is defined at a node (see *Transformed Coordinate Systems*); otherwise, they are applied in global directions.

Applying Lagrangian Mesh Constraints

Lagrangian mesh constraints on a node are used to indicate that mesh smoothing should not be applied; i.e., the node must follow the material.

Input File Usage: **ADAPTIVE MESH CONSTRAINT*,
CONSTRAINT TYPE=LAGRANGIAN

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Follow underlying material**

Modifying ALE Adaptive Mesh Constraints

By default, all adaptive mesh constraints defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh constraints in effect for a given step relative to the preexisting adaptive mesh constraints. At each new step the existing adaptive mesh constraints can be modified and additional adaptive mesh constraints can be specified.

Input File Usage: Use either of the following options to modify an existing adaptive mesh constraint or to specify an additional adaptive mesh constraint:

**ADAPTIVE MESH CONSTRAINT*,
**ADAPTIVE MESH CONSTRAINT*, OP=MOD

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Manager**: select the desired step and mesh constraint: **Edit**

Removing ALE Adaptive Mesh Constraints

If you choose to remove any adaptive mesh constraint in a step, no adaptive mesh constraints will be propagated from the previous step. Therefore, all adaptive mesh constraints that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh constraints and to specify new adaptive mesh constraints:

**ADAPTIVE MESH CONSTRAINT*, OP=NEW

If the OP=NEW parameter is used on any **ADAPTIVE MESH CONSTRAINT* option within a step, it must be used on all **ADAPTIVE MESH CONSTRAINT* options in the step.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Manager**: select the desired step and mesh constraint: **Deactivate**

Initial Conditions

There are no initial conditions specific to adaptive meshing; initial conditions can be defined in the same way as in nonadaptive problems. If initial mesh sweeps are performed to smooth the mesh at the beginning of a step (see *ALE Adaptive Meshing and Remapping in Abaqus/Explicit*), all initial conditions (except temperatures and field variables, which are discussed in *Predefined Fields*, presented later in this section) are remapped to the new mesh. Initial temperatures are remapped during adaptive meshing in an adiabatic analysis.

Initial conditions prescribed near an inflow Eulerian boundary region will affect the state of the material flowing into the domain throughout the analysis. See *Modeling Techniques for Eulerian Adaptive Mesh Domains in Abaqus/Explicit* for a discussion of the proper treatment of inflow boundaries.

Defining Surfaces on ALE Adaptive Mesh Boundaries

When you define a surface on the boundary of an adaptive mesh domain (see *About Surfaces*), Abaqus creates a boundary region coinciding with the surface. By default, a sliding boundary region is created. You can choose to create a Lagrangian or Eulerian boundary region instead.

A surface defined in the interior of an adaptive mesh domain will move independently of the material (unless constrained by mesh constraints).

Defining a Sliding Boundary Region Using a Surface

By default, the boundary region created by a surface definition will be sliding (the surface edge will slide freely over the material).

Input File Usage: **SURFACE*, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Defining a Lagrangian Boundary Region Using a Surface

To force the surface edge to follow the material, create a Lagrangian boundary region.

Input File Usage: **SURFACE*, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Defining an Eulerian Boundary Region Using a Surface

To decouple the surface from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the surface will behave like a sliding boundary region (no material will flow through the surface).

As an example, it is often assumed that there is no normal or shear stress in the material at the outflow boundary of an Eulerian domain. This condition can be modeled by defining an Eulerian boundary region using a surface and applying spatial mesh constraints perpendicular to the surface, as shown in *Figure 5*.

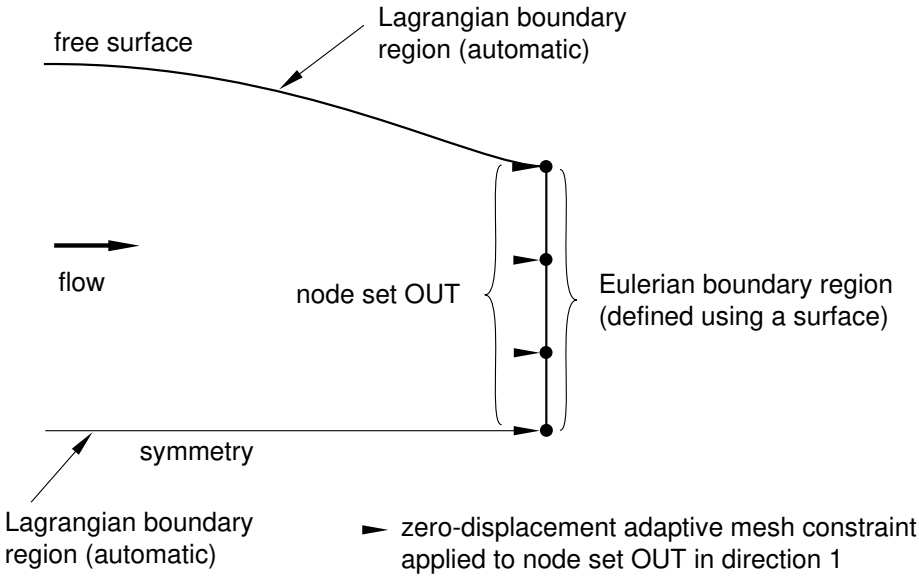


Figure 5: Modeling the outflow boundary of an Eulerian adaptive mesh domain.

Input File Usage: **SURFACE*, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Contact

Lagrangian and sliding boundary regions created using surfaces can be used in general contact and contact pairs; they have the same meaning as surfaces defined on nonadaptive regions. Since contact generally involves relative sliding between bodies, sliding boundary regions are typically appropriate for contact surfaces.

Surfaces defined on Eulerian boundary regions cannot be used in the contact definition.

The nodes of an element-based surface in a no-separation contact interaction are nonadaptive (see *Contact Pressure-Overclosure Relationships*)

The following restrictions are applied only when the contact pair algorithm is used:

- Nodes that belong to an adaptive region cannot be part of a contact pair with a self-contact definition. If the nodes are part of a contact pair with a self-contact definition, contact is not enforced.
- If the small-sliding formulation is used for a contact pair, all the nodes on both surfaces are nonadaptive (see *About Contact Pairs in Abaqus/Explicit* and *Contact Formulations for Contact Pairs in Abaqus/Explicit*).

Distributed Loads

When a distributed pressure load is applied to the boundary of an adaptive mesh domain, Abaqus/Explicit creates a boundary region that coincides with the area of the load application. The characteristics of boundary regions created in this way are identical to those of boundary regions created by defining surfaces. If a pressure load is applied to a surface in the interior of an adaptive mesh domain, the nodes on the surface will move with the material in all directions (i.e., they will be nonadaptive).

Boundary regions created by different pressure loads may overlap. If pressure loads with the same magnitude and amplitude definition are applied to adjacent regions, the regions will be merged into a single boundary region to minimize the number of Lagrangian edges and corners formed (see *Figure 6*).

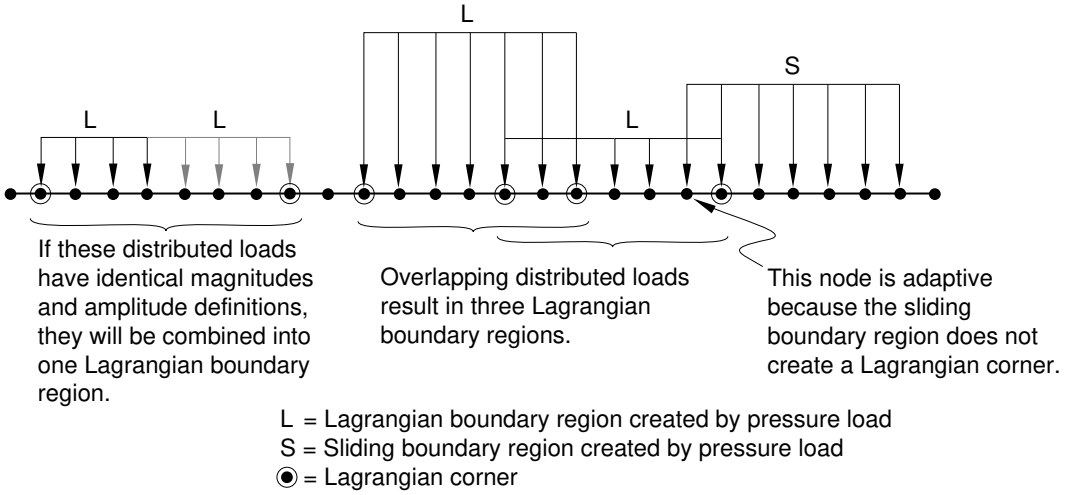


Figure 6: Applying distributed pressure loads to an adaptive mesh domain.

If a nonuniform pressure is applied (for example, a pressure that varies linearly over a surface) or if a pressure load is defined in user subroutine *VDLOAD*, each element face or edge becomes a separate Lagrangian boundary region. Since Lagrangian corners are formed where Lagrangian edges meet, all nodes will follow the material in every direction, and each region becomes nonadaptive. Likewise, if a nonuniform body force is applied to an adaptive mesh domain, the domain is split into multiple domains, each with a uniform body force. If this splitting results in one-element domains, the region becomes nonadaptive.

Defining a Lagrangian Boundary Region with a Pressure Load

By default, the boundary region created to coincide with a pressure load will be Lagrangian. Pressure loads applied to Lagrangian regions are identical to pressure loads applied to nonadaptive regions, except that the mesh can move inside the boundary region.

Input File Usage: **DLOAD*, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

Defining a Sliding Boundary Region with a Pressure Load

A pressure load can be applied to a sliding boundary region to simulate a load that is fixed in space while material moves past it (see *Figure 7*). A sliding edge is unconstrained in the direction tangential to the boundary region; therefore, unless adaptive mesh constraints are applied, the area of the load application will move according to the adaptive meshing algorithm, which is probably not physically meaningful.

To allow a pressure load to slide over the material, create a sliding boundary region.

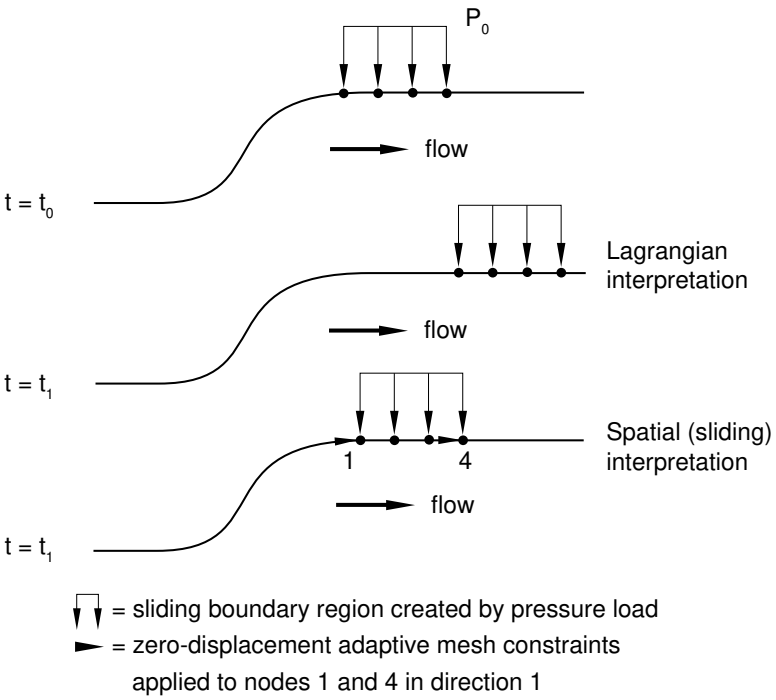


Figure 7: Applying a sliding distributed pressure load to an adaptive mesh domain.

Input File Usage: **DLOAD*, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

Defining an Eulerian Boundary Region with a Pressure Load

To decouple the area of pressure application from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the mesh will behave like a sliding boundary region (no material will flow through the surface).

As an example, it is often assumed that a uniform ambient pressure exists at the outflow boundary of an Eulerian domain. This condition can be modeled by defining the pressure at an Eulerian boundary region using a distributed load and applying spatial mesh constraints perpendicular to the surface, as shown in *Figure 8*.

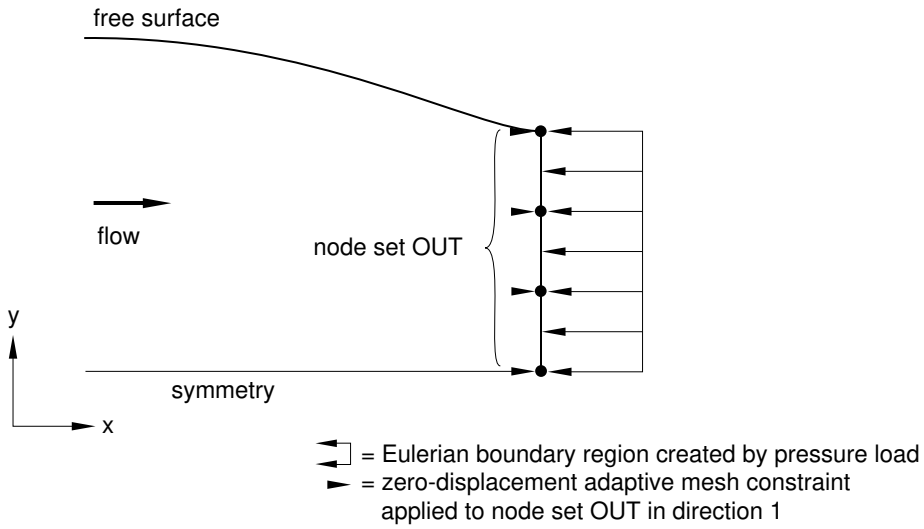


Figure 8: Modeling an ambient pressure at the outflow boundary of an Eulerian adaptive mesh domain.

Input File Usage: **DLOAD*, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

Distributed Surface Fluxes and Thermal Conditions

In coupled thermal-stress analysis Abaqus/Explicit also creates boundary regions for distributed surface fluxes, convective film conditions, and radiation conditions. The rules governing boundary regions for these loads are identical to those discussed for distributed loads. The ability to specify the boundary region type is also the same.

Concentrated Loads

When a concentrated load is applied to the boundary of an adaptive mesh domain, Abaqus/Explicit creates a boundary region to coincide with the load. Every node to which a concentrated load is applied will be considered its own boundary region because it is not possible to identify a surface area associated with a concentrated load. However, you can control the behavior of each one-node region.

If concentrated loads are applied to nodes in the interior of an adaptive mesh domain, those nodes will move with the material in all directions (i.e., they will be nonadaptive).

Defining a Lagrangian Boundary Region with a Concentrated Load

By default, the boundary region created by a concentrated load will be Lagrangian. Each one-node Lagrangian boundary region will follow the material in every direction (the nodes will be nonadaptive).

Input File Usage: **CLOAD*, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Defining a Sliding Boundary Region with a Concentrated Load

A concentrated load can be applied to a sliding boundary region to simulate a load that is fixed in space while material moves past it (see *Figure 9*).

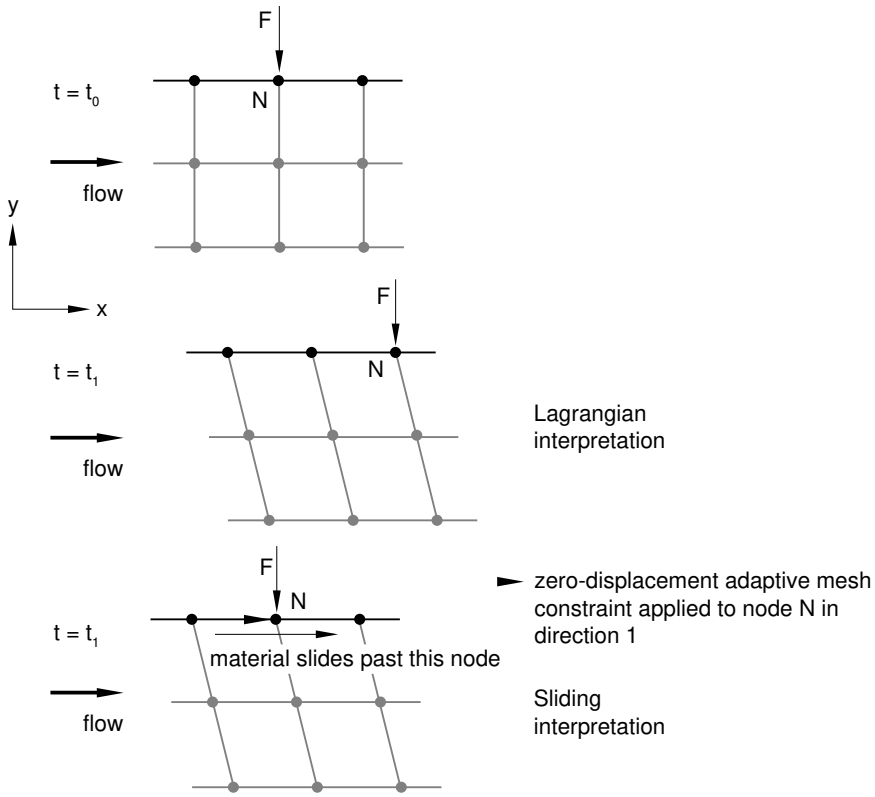


Figure 9: Applying a concentrated sliding load to an adaptive mesh domain.

A sliding node is unconstrained in the direction tangential to the boundary region; therefore, unless adaptive mesh constraints are applied, the point of load application will move according to the adaptive meshing algorithm, which is probably not physically meaningful.

To allow the concentrated load to slide freely over the material, create a sliding boundary region.

Input File Usage: **CLOAD*, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Defining an Eulerian Boundary Region with a Concentrated Load

To decouple the concentrated load from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, each one-node boundary region will behave like a sliding boundary region.

Input File Usage: **CLOAD*, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Concentrated Fluxes and Thermal Conditions

In coupled thermal-stress analysis Abaqus/Explicit also creates boundary regions for concentrated heat fluxes, film conditions, and radiation conditions. The rules governing boundary regions for these loads are identical to those discussed for concentrated loads. The ability to specify the boundary region type is also the same.

Boundary Conditions

Lagrangian, sliding, and Eulerian boundary regions can be created by applying kinematic constraints to the boundary of an adaptive mesh domain. If kinematic boundary conditions are applied to nodes in the interior of an adaptive mesh domain, those nodes will move with the material in all directions (i.e., they will be nonadaptive), regardless of the specified boundary region type.

Defining a Lagrangian Boundary Region Using a Boundary Condition

By default, the boundary region created by a kinematic boundary condition will be Lagrangian. Abaqus/Explicit will recognize surface-type and point or edge constraints automatically and will create an appropriate Lagrangian boundary region for each type, as explained in the following subsections.

Input File Usage: **BOUNDARY*, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Surface-Type Constraints Applied Using Boundary Conditions

Although boundary conditions are always applied to individual nodes in Abaqus/Explicit, they often represent physical constraints on surfaces. For example, symmetry conditions, where nodes are constrained to move in a plane, are actually surface constraints. A fully clamped (ENCASTRE) condition along a boundary can also be considered a surface constraint. (During adaptive meshing it is meaningful to allow nodes to move along a fully clamped edge.)

Abaqus/Explicit will examine an adaptive mesh boundary and try to create regions that are coincident with the applied boundary conditions. Currently, Abaqus/Explicit can create boundary regions for surface-based constraints on:

- symmetry planes,
- fully clamped planes, and
- planes on which a uniform motion is prescribed.

Figure 2 shows an example in which boundary regions are created by applying surface-type boundary conditions. This figure shows a block of material with a crack and three symmetry planes (therefore, three Lagrangian boundary regions). Material will not flow across any symmetry plane, yet adaptive meshing can be performed within each Lagrangian boundary region. This flexibility is often helpful in problems that have significant deformation.

Point or Edge Constraints Applied Using Boundary Conditions

Some boundary conditions represent point or edge constraints. For example, a displacement can be prescribed at a node. The boundary regions associated with such nodes are exactly the same as those created by concentrated loads.

Defining a Sliding Boundary Region Using a Boundary Condition

A sliding boundary region associated with a boundary condition can move according to the adaptive meshing algorithm. Since this behavior is probably not physically meaningful, the edges of a sliding boundary region are usually fixed in the direction tangential to the surface using adaptive mesh constraints. This approach can be used, for example, to simulate frictionless contact between a rigid punch and a deformable body, as shown in *Figure 10*.

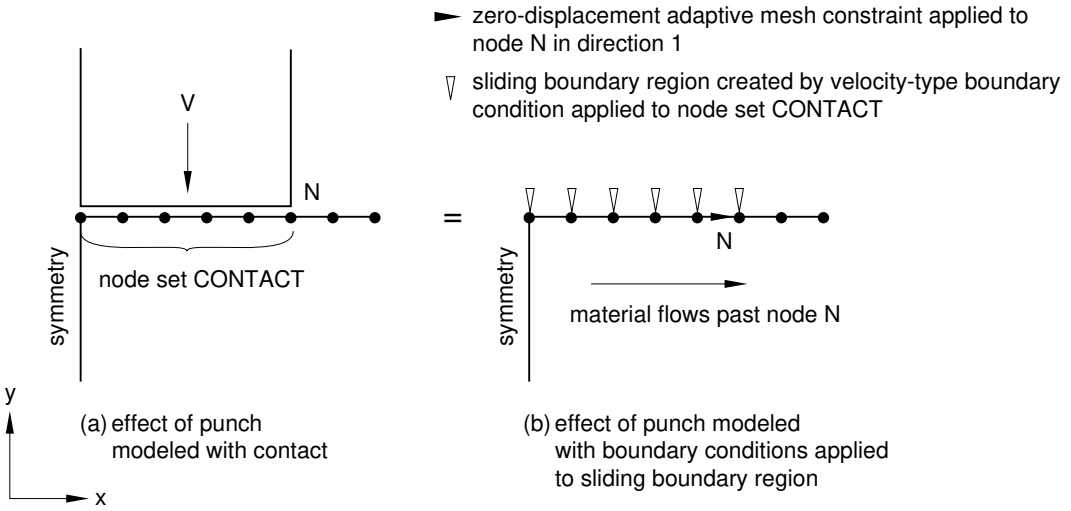


Figure 10: Contact simulation using a sliding boundary region.

In this example the punch is replaced by a sliding boundary region with a constant velocity boundary condition applied in the area of "contact." A tangential mesh constraint is applied to the edge of the boundary region at node N (the other edge is constrained by the Lagrangian boundary region created automatically on the symmetry plane). This problem definition allows material to flow radially underneath the "punch" while retaining the original size and location of the "contact" area.

Abaqus/Explicit makes no distinction between surface-type constraints and point or edge constraints for sliding boundary regions.

To allow the boundary condition to slide freely over the material, create a sliding boundary region.

Input File Usage: **BOUNDARY*, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Defining an Eulerian Boundary Region Using a Boundary Condition

To decouple the boundary region from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the mesh will behave like a sliding boundary region (no material will flow through the surface).

As an example, the mass flow rate at an Eulerian inflow boundary can be prescribed by defining an Eulerian boundary region using a boundary condition.

Abaqus/Explicit makes no distinction between surface-type constraints and point or edge constraints for Eulerian boundary regions.

Input File Usage: **BOUNDARY*, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Overlapping Boundary Regions

A Lagrangian boundary region can overlap any number of other Lagrangian or sliding boundary regions (see *Figure 11*). If two boundary regions partially overlap, three regions are formed: the overlapping region and the two initial regions minus the overlapping region. A sliding boundary region is formed when a Lagrangian and a sliding boundary region overlap.

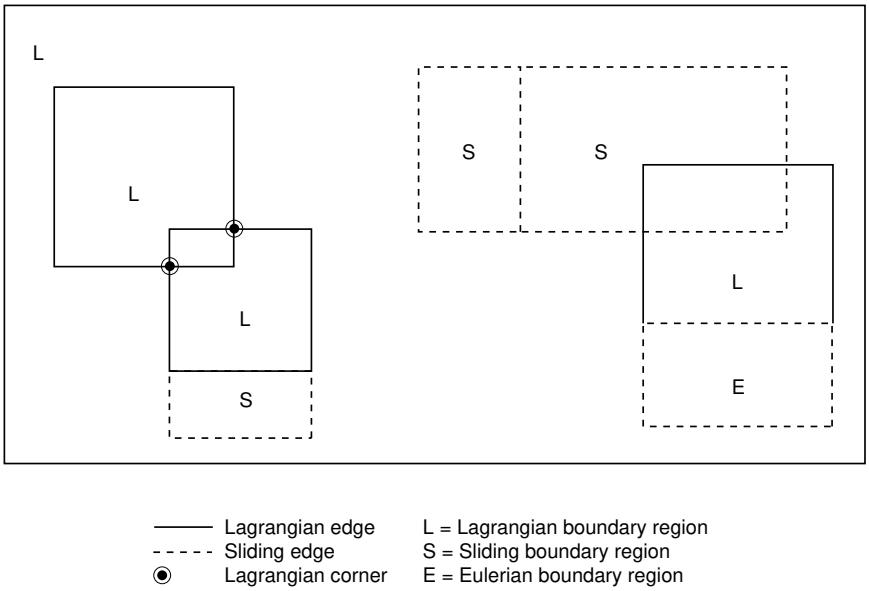


Figure 11: Overlapping boundary regions.

An Eulerian boundary region can never overlap a Lagrangian or sliding boundary region. Furthermore, an Eulerian boundary region can never share a boundary with or overlap a nonadaptive region. Because infinite elements are nonadaptive, this latter restriction implies that infinite elements cannot be used to simulate ambient conditions at an outflow boundary.

Coincident Edges

Edges shared by different types of boundary regions are subject to the following rules:

- An edge shared between a Lagrangian and a sliding boundary region will be Lagrangian.
- An edge shared between a Lagrangian and an Eulerian boundary region will be sliding.
- An edge shared between a Lagrangian and a nonadaptive boundary region will be nonadaptive.
- An edge shared between a sliding and a nonadaptive boundary region will be nonadaptive.
- An edge of an Eulerian boundary region can never be coincident with an edge of a nonadaptive region.

Predefined Fields

There are no restrictions on applying prescribed temperatures or field variables in an adaptive mesh domain, but these nodal values are not remapped when adaptive meshing is performed. Therefore, predefined fields that are not spatially uniform may not be meaningful within an adaptive mesh domain. (Time-varying, spatially uniform predefined fields are acceptable, since adaptive meshing is applied at discrete instances in time.) However, if temperature or field variable data are collected from a spatial frame of reference, it may make physical sense to apply a spatially varying field for an Eulerian domain in which the mesh does not move. Abaqus/Explicit does not perform any checks or calculations on predefined fields for adaptive meshing; you must ensure that the predefined fields are meaningful.

Materials

All material models and behaviors, except brittle cracking (*Cracking Model for Concrete*) and fabric (*Fabric Material Behavior*) materials, can be used in an adaptive mesh domain.

For domains modeled with hyperelastic or hyperfoam materials the usefulness of adaptive meshing is limited. The recommended enhanced hourglass method (*Section Controls*), which will generally predict a much better return to the original configuration for these materials when loading is removed, cannot be used in an adaptive mesh domain. Therefore, for hyperelastic or hyperfoam materials it is recommended that the analysis be run without adaptive meshing but with enhanced hourglass control.

If the porous failure model (*Failure Criteria in Abaqus/Explicit*), shear failure model (*Shear Failure Model*), tensile failure model (*Tensile Failure Model*), or one of the progressive damage models (*Progressive Damage and Failure*) is specified within an adaptive mesh domain, Abaqus/Explicit will continuously monitor the status of elements while performing adaptive meshing. When elements within the domain fail, the nodes along the interface between the failed and unfailed elements will become nonadaptive. This has the effect of creating a material boundary between the failed and unfailed zones.

When failure occurs in elements that use the shear failure, the tensile failure, or the progressive damage models without element deletion, elements in the failure zone will not be deleted; they can carry some states of stress. Adaptive meshing will occur within the failure zone but not along the interface with the unfailed material.

Elements

An adaptive mesh domain can contain only first-order, reduced-integration, solid elements. All elements within an adaptive mesh domain must have the same geometry (all two-dimensional, three-dimensional, axisymmetric, or plane strain, etc.). Since adaptive mesh domains are split across element types, degenerate elements should be used for mixed domains that include both triangles and quadrilaterals (or tetrahedron and bricks). All elements other than first-order, reduced-integration, solid elements—including mass, rotary inertia, and infinite elements—are nonadaptive. These elements can be connected to an adaptive mesh domain, but their nodes are nonadaptive. All nodes and elements on a rigid body are nonadaptive. Rebar are not supported within an adaptive mesh domain.

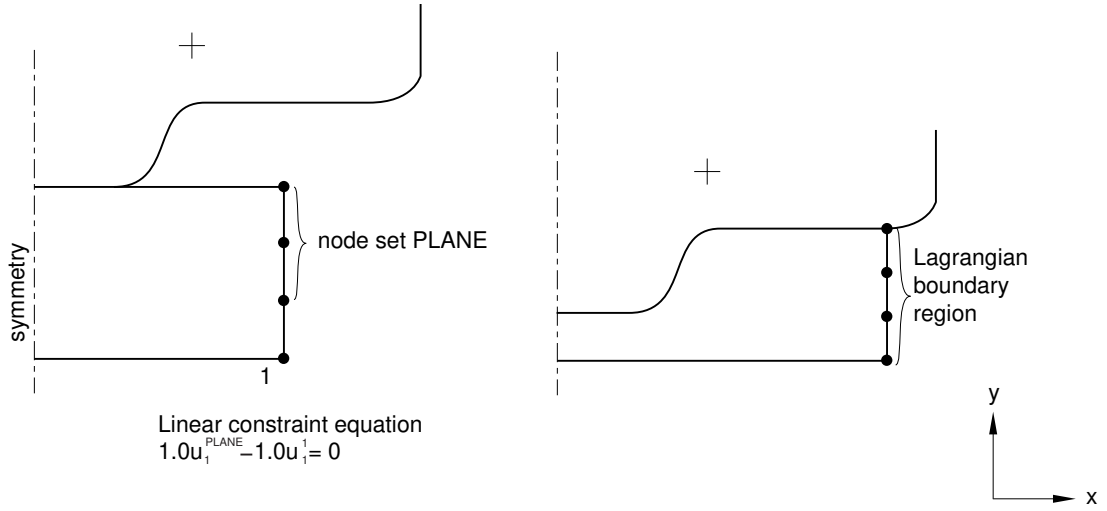
Multi-Point Constraints and Equations

As with boundary conditions, multi-point constraints (*General Multi-Point Constraints*) and equations (*Linear Constraint Equations*) are always applied to nodes but sometimes represent constraints on surfaces. Abaqus/Explicit will recognize surface-type constraints when the following conditions are satisfied:

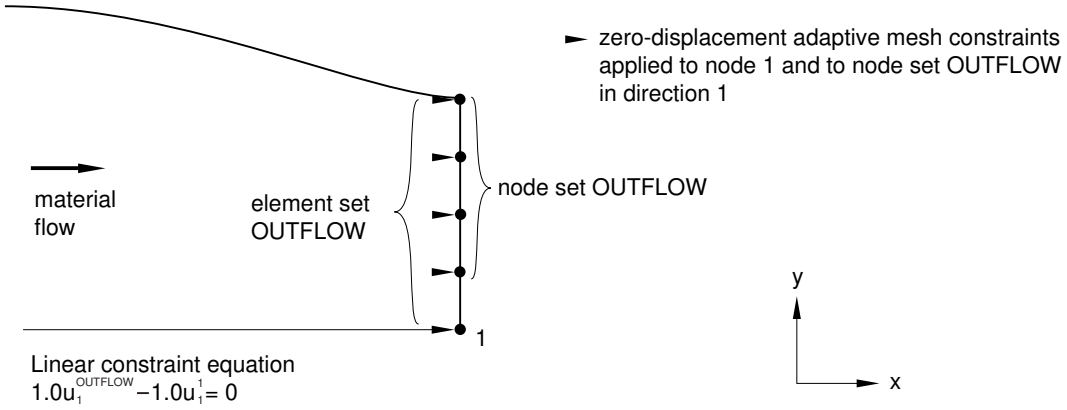
- an equation, PINMPC, or TIEMPC ties a node set to a single node; and
- all the nodes involved in the MPC or equation are coplanar and lie within the boundary region.

If these conditions are satisfied, a boundary region will be associated with the node set in the MPC or equation. If the MPC is applied within a Lagrangian or sliding boundary region, a Lagrangian edge will be created. If the MPC is applied within an Eulerian boundary region, no edge will be created. If the above conditions are not satisfied, all nodes connected to the MPC or equation will be nonadaptive.

As an example, a constraint can be applied to force a plane section to remain plane in a Lagrangian adaptive mesh domain, as shown in *Figure 12*(a). In this case all nodes are constrained by an equation to lie in the same plane throughout the analysis, but adaptive meshing is allowed within the plane.



(a) Using an equation to force a plane section to remain a plane.



(b) Using an equation to prescribe a uniform velocity outflow condition.

Figure 12: Using equations with adaptive meshing.

As another example, consider the outflow boundary of an Eulerian domain, as shown in *Figure 12*(b). The outflow boundary of an Eulerian domain is often assumed to be far enough downstream that the velocity is uniform but unknown. To model this condition, an Eulerian boundary region is created at the outflow boundary using a surface. An adaptive mesh constraint is used to fix the mesh perpendicular to the boundary, and all nodes on the plane are constrained by an equation to have the same velocity orthogonal to the plane.

For surface-based tie constraints (see *Mesh Tie Constraints*), all nodes on the tied surfaces will be nonadaptive.

Procedures

During an adiabatic analysis temperatures will be remapped properly in adaptive mesh domains. Adaptive meshing is not used during annealing procedures or during geometrically linear analyses.

The definitions of adaptive mesh domains, boundary regions, mesh constraints, and controls (as explained in *ALE Adaptive Meshing and Remapping in Abaqus/Explicit*) will propagate from step to step.

User Subroutines

Solution-dependent state variables defined in user subroutine *VUMAT* will be remapped to the new mesh when adaptive meshing is performed.

Solution-dependent state variables that are defined on a secondary surface in user subroutines *VFRIC*, *VUINTER*, *VFRICTION*, and *VUINTERACTION* will not be remapped to the new mesh when adaptive meshing is performed. Therefore, to ensure physically meaningful results, a Lagrangian adaptive mesh constraint should be used for nodes on the contact secondary surfaces with solution-dependent state variables where the contact constraint is defined using these user subroutines.

Output

Since the mesh is no longer constrained to the material when adaptive meshing is performed, output at elements and nodes must be interpreted differently than in a pure Lagrangian problem. See *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit* for details.

Input File Template

To create a Lagrangian adaptive mesh domain:

```
*HEADING
 …
*ELSET, ELSET=ADAPT
************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
*ADAPTIVE MESH, ELSET=ADAPT
...
*END STEP
```

To create an Eulerian adaptive mesh domain with a prescribed velocity inflow condition and a prescribed pressure outflow condition (both in the global *x*-direction):

```
*HEADING...
*ELSET, ELSET=ADAPT
...
*ELSET, ELSET=OUT
...
*NSET, NSET=INFLOW
...
*NSET, NSET=OUTFLOW
...
*SURFACE, NAME=INSURF, REGION TYPE=EULERIAN
Data lines to define the surface
*SURFACE, NAME=OUTSURF, REGION TYPE=EULERIAN
Data lines to define the surface
...
*EQUATION
Data lines specifying uniform velocity at the inflow
************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
*ADAPTIVE MESH, ELSET=ADAPT
*ADAPTIVE MESH CONSTRAINT
 INFLOW,  1, 1, 0
 OUTFLOW, 1, 1, 0
```

```
*BOUNDARY, TYPE=VELOCITY, REGION TYPE=EULERIAN
 INFLOW, 1, 1, 10.0
*DLOAD, REGION TYPE=EULERIAN
 OUT, P2, 15.0
...
*END STEP
```

ALE Adaptive Meshing and Remapping in Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References:

- *About ALE Adaptive Meshing*
- *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*
- *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*
- **ADAPTIVE MESH*
- **ADAPTIVE MESH CONSTRAINT*
- **ADAPTIVE MESH CONTROLS*
- *Customizing ALE adaptive meshing*

Overview

ALE adaptive meshing consists of two fundamental tasks:

- creating a new mesh, and
- remapping solution variables from the old mesh to the new mesh with a process called advection.

The success of the adaptive meshing technique depends on the choice of the methods used for each of these tasks. The default methods for creating a new mesh and for remapping solution variables have been chosen carefully to work for a wide variety of problems. However, you might wish to override the default choices to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing to more difficult or unusual applications.

Meshing

A new mesh:

- is created at a specified frequency for each adaptive domain;
- is found by sweeping iteratively over the adaptive mesh domain and moving nodes to smooth the mesh; and
- can retain the initial gradation of the original mesh.

Remapping

The methods used for advecting solution variables to the new mesh:

- are consistent, monotonic, and (by default) accurate to the second order; and
- conserve mass, momentum, and energy.

Controlling the Frequency of ALE Adaptive Meshing

In most cases the frequency of adaptive meshing is the parameter that most affects the mesh quality and the computational efficiency of adaptive meshing. A typical adaptive mesh application without Eulerian boundaries will require adaptive meshing every 5–100 increments. In contrast, adaptive meshing should generally be performed much more frequently in a steady-state process simulation using Eulerian boundaries. Thus, if a spatial adaptive mesh constraint or an Eulerian boundary region is defined on an adaptive mesh domain, the default frequency is 1; otherwise, the default frequency is 10.

Input File Usage: Use the following option to change the frequency of adaptive meshing:

**ADAPTIVE MESH*, FREQUENCY=*number of increments*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Frequency:** *number of increments*

Controlling the Intensity of ALE Adaptive Meshing

During each adaptive meshing increment, the new mesh is created by performing one or more mesh sweeps and then advecting the solution variables to the new mesh.

Mesh Sweeps

In an adaptive meshing increment, a new, smoother mesh is created by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated—based on the current positions of neighboring nodes and elements—to reduce element distortion. In a typical sweep a node is moved a fraction of the characteristic length of any element surrounding the node. Increasing the number of sweeps increases the intensity of adaptive meshing in each adaptive meshing increment. The default number of mesh sweeps is one.

Input File Usage: Use the following option to change the number of mesh sweeps to be performed in each adaptive mesh increment:

**ADAPTIVE MESH*, MESH SWEEPS=*number of sweeps*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Remeshing sweeps per increment:** *number of sweeps*

Advection Sweeps

The process of mapping solution variables from an old mesh to a new mesh is referred to as an advection sweep. At least one advection sweep is performed in every adaptive mesh increment. Ideally, an advection sweep will be performed only once, after all mesh sweeps for the increment are complete. However, numerical stability of the advection sweep is maintained only if the difference between the old mesh and the new mesh is small. Therefore, if after a mesh sweep the total accumulated movement of any node in the domain is greater than 50% of the characteristic length of any adjacent element, an advection sweep is performed to remap the solution variables from the old mesh to the intermediate mesh. Mesh sweeps will continue until the specified number is reached or until the movement of any node again exceeds the 50% threshold. At this time an advection sweep is performed again to map variables from the last intermediate mesh to the new intermediate mesh. The cycle will continue until the number of mesh sweeps reaches the specified number.

The number of advection sweeps per adaptive mesh increment required for each adaptive mesh domain is determined automatically by Abaqus/Explicit; you cannot override this automatic calculation. The number of advection sweeps is printed by default to the message (`.msg`) file (see *Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit*).

The Computational Cost of ALE Adaptive Meshing

The cost of adaptive meshing depends on the frequency of remeshing, the number of mesh and advection sweeps performed, and the size of the adaptive mesh domains. When compared to a purely Lagrangian analysis, additional computational cost is incurred only within adaptive mesh increments.

Generally, the cost of one advection sweep is several times greater than the cost of one mesh sweep. Multiple advection sweeps are triggered when adaptive meshing is performed too infrequently and/or a high number of mesh sweeps is specified. Performing adaptive meshing more frequently and doing 1–5 mesh sweeps in each adaptive mesh increment will usually generate only one advection sweep, minimizing the computational cost.

The relatively smooth mesh and improved element aspect ratios that result from adaptive meshing may increase the stable time increment compared to a similar pure Lagrangian analysis. In some cases this increase can offset the cost of adaptive meshing completely.

Although computational cost can vary greatly with the type of application, performing adaptive meshing on the entire problem domain in every increment will typically increase the cost of the analysis by 3–5 times that of a similar Lagrangian analysis. Defining adaptive mesh domains that cover only a fraction of the entire problem domain will reduce the cost proportionally. Changing the frequency to every 10–25 increments will result in CPU times that are only moderately higher than those for a pure Lagrangian analysis.

Guidelines for Controlling ALE Adaptive Meshing Frequency and Intensity

Although the default values work well for many problems, difficult analyses may require a more frequent adaptive meshing frequency or meshing with a higher intensity.

Guidelines for Transient Analysis

For problems without spatial adaptive mesh constraints or Eulerian boundary regions, the default frequency for adaptive meshing is 10, and the default number of mesh sweeps is 1. The default values are usually adequate for low- to moderate-rate dynamic problems and for quasi-static process simulations undergoing moderate deformation. If the frequency or number of mesh sweeps is too low, excess element distortion may cause the analysis to terminate before the mesh is adapted; or, if a solution can be obtained, it may not be as accurate as the solution that could be obtained with a higher quality mesh. In virtually all cases, however, performing adaptive meshing at any frequency will reduce the distortion of elements (and, thus, improve the quality of the solution) compared to a pure Lagrangian analysis.

For high-rate impact problems undergoing large amounts of deformation, it might be necessary to increase the frequency of adaptive meshing or the number of mesh sweeps. It is generally less expensive to increase the number of mesh sweeps slightly before increasing the frequency, as long as the number of advection sweeps remains small.

For problems involving explosions taking place over just a few hundred increments, adaptive meshing is usually required at every increment. It may also be necessary to increase the frequency of adaptive meshing for quasi-static process simulations that involve large amounts of flow per increment.

For problems in which the deformation per increment is small, a high-quality mesh can be maintained by performing adaptive meshing only every 25–100 increments. For these problems the additional cost of adaptive meshing is negligible.

Guidelines for Steady-State Analysis

When an adaptive mesh domain contains Eulerian boundary regions or has spatial adaptive mesh constraints, the default frequency of adaptive meshing is 1. This default frequency is conservative and is chosen primarily because spatial mesh constraints are applied only during adaptive mesh increments. Thus, between adaptive mesh increments the mesh may drift from its prescribed location, which may affect the solution. However, drift from adaptive mesh constraints will always be eliminated in the next adaptive mesh increment: it will not accumulate.

For problems in which the speed of deformation or the speed of material flow from element to element is much less than the material wave speed, the frequency typically can be increased to 5 or higher. This class of problems includes most steady-state process simulations, where the drift of the mesh from the prescribed location is negligible over a few increments. By performing adaptive meshing less often, steady-state simulations become

competitive with their corresponding transient simulations. For Eulerian domains in which the speed of the deformation or material flow is high, such as in dynamic shock problems, the default frequency of 1 should be used.

Mesh Smoothing Methods

The determination of the new mesh in Abaqus/Explicit is based on four aspects. You can control each of these aspects by defining adaptive mesh controls. Defaults have been chosen so that the overall algorithm works well for most problems.

First, the calculation of the new mesh in Abaqus/Explicit is based on some combination of three basic smoothing methods: volume smoothing, Laplacian smoothing, and equipotential smoothing. The smoothing methods are applied at each node in the adaptive mesh domain to determine the new location of the node based on the locations of surrounding nodes or elements. Although all the smoothing methods tend to smooth the mesh and reduce element distortion, the resulting meshes will differ depending on the methods used.

Second, initial element gradation can be maintained at the expense of element distortion if desired. Third, optimal positioning of the nodes before the basic smoothing methods are applied can improve mesh quality and minimize the frequency of adaptive meshing required.

Finally, solution-dependent meshing is used to concentrate mesh refinement near areas of evolving boundary curvature. This counteracts the tendency of the basic smoothing methods to reduce the mesh refinement near concave boundaries where solution accuracy is important.

Volume Smoothing

Volume smoothing relocates a node by computing a volume-weighted average of the element centers in the elements surrounding the node. In *Figure 1* the new position of node M is determined by a volume-weighted average of the positions of the element centers, C, of the four surrounding elements. The volume weighting will tend to push the node away from element center C1 and toward element center C3, thus reducing element distortion.

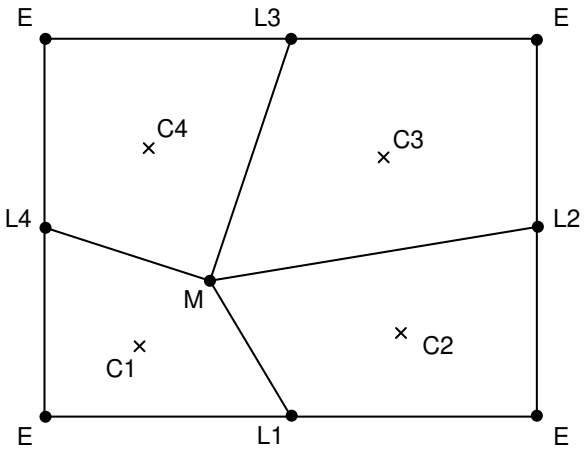


Figure 1: Relocation of a node during a mesh sweep.

Volume smoothing is very robust and is the default method in Abaqus/Explicit. It works well for both structured and highly unstructured domains. (A structured domain is one that contains no degenerate elements and where every node is surrounded by four elements in two dimensions or eight elements in three dimensions.)

Laplacian Smoothing

Laplacian smoothing relocates a node by calculating the average of the positions of each of the adjacent nodes connected by an element edge to the node in question. In *Figure 1* the new position of node M is determined by

averaging the positions of the four nodes, L, connected to M by element edges. The locations of nodes L2 and L3 will pull node M up and to the right to reduce element distortion.

Laplacian smoothing is the least expensive smoothing algorithm and is commonly used in mesh preprocessors. For low to moderately distorted mesh domains, the results of Laplacian smoothing are similar to volume smoothing. For domains with boundaries of complex curvature, volume smoothing generally results in a more balanced mesh.

Equipotential Smoothing

Equipotential smoothing is a higher-order method that relocates a node by calculating a higher-order, weighted average of the positions of the node's eight nearest neighbor nodes in two dimensions (or its eighteen nearest neighbor nodes in three dimensions). In *Figure 1* the new position of node M is based on the position of all the surrounding nodes, L and E.

The weighted averaging for the equipotential smoothing method is fairly complex and is based on the solution of the Laplace equation. Equipotential smoothing tends to minimize the local curvature of lines running across a mesh over several elements. Although this tendency can be desirable for gently curving domains, it can inhibit the ability of equipotential smoothing to reduce element distortion in highly deformed and locally curved domains.

Equipotential smoothing can be performed only for nodes that are surrounded by a locally structured mesh. Nodes that are surrounded by an unstructured mesh are moved with an equivalent amount of volume smoothing when equipotential smoothing is chosen.

Combining Smoothing Methods

The default smoothing method in Abaqus/Explicit is volume smoothing. To choose an alternate smoothing method or to combine smoothing methods, you specify the weighting factor for each method. When more than one smoothing method is used, a node is relocated by computing a weighted average of the locations predicted by each chosen method. All weights must be positive, and their sum should typically be 1.0. If the sum of the chosen weights is less than 1.0, the mesh smoothing algorithm will be less aggressive at each adaptive mesh increment. If the sum of the chosen weights is greater than 1.0, their values are normalized so that their sum is 1.0.

Input File Usage:

**ADAPTIVE MESH CONTROLS*, NAME=*name*
volume smoothing weight, Laplacian smoothing weight, equipotential smoothing weight

For example, the following option could be used to define an equal blend of volume and equipotential smoothing, with no Laplacian smoothing:

**ADAPTIVE MESH CONTROLS*, NAME=*name*
0.5, 0.0, 0.5

Abaqus/CAE Usage:

Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Volumetric:** *volume smoothing weight*, **Laplacian:** *Laplacian smoothing weight*, **Equipotential:** *equipotential smoothing weight*

Geometric Enhancements to the Basic Smoothing Methods

The conventional forms of the basic smoothing methods do not perform well in highly distorted domains. To ensure the robustness of adaptive meshing, Abaqus/Explicit uses geometrically enhanced forms of the basic smoothing algorithms by default. The enhanced forms are recommended for all adaptive mesh applications. However, since the basic smoothing algorithms are used by many finite element preprocessors, their conventional forms are provided as an option.

Input File Usage: Use the following option to use the conventional forms of the volume, Laplacian, or equipotential smoothing algorithms:

**ADAPTIVE MESH CONTROLS*, NAME=*name*,
GEOMETRIC ENHANCEMENT=NO

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, toggle off **Use enhanced algorithm based on evolving element geometry**

Specifying a Uniform Mesh Smoothing Objective

For adaptive mesh domains without any Eulerian boundary regions, the default objective of the mesh smoothing methods is to minimize mesh distortion while improving element aspect ratios, at the expense of diffusing initial mesh gradation. The uniform mesh smoothing objective is recommended for problems with moderate to large overall deformation.

Input File Usage: Use the following option to specify the uniform mesh smoothing objective:

**ADAPTIVE MESH CONTROLS*, NAME=*name*,
SMOOTHING OBJECTIVE=UNIFORM

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Priority: Improve aspect ratio**

Specifying a Graded Mesh Smoothing Objective

Alternatively, the smoothing methods can attempt to preserve initial mesh gradation while reducing element distortion as the analysis evolves. This objective is the default for adaptive mesh domains with one or more Eulerian boundary regions. The graded mesh smoothing objective is recommended only for adaptive mesh domains with reasonably structured graded meshes undergoing low to moderate overall deformation. Element distortion will be minimized, but the aspect ratios of adjacent elements will be maintained approximately. Mesh gradation is particularly useful in steady-state problems where overall deformations are small and a focused mesh is used in a specific area to capture high solution gradients.

Input File Usage: Use the following option to specify the graded mesh smoothing objective:

**ADAPTIVE MESH CONTROLS*, NAME=*name*,
SMOOTHING OBJECTIVE=GRADED

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Priority: Preserve initial mesh grading**

Positioning Nodes in Lagrangian Domains

If an adaptive mesh domain has no Eulerian boundary regions, then, by default, the mesh sweeps are based on current nodal locations, which account for material motion accumulated since the last adaptive mesh increment. This approach is generally the best for Lagrangian problems that undergo large overall deformation.

Input File Usage: Use the following option to request that the current deformed positions of nodes be used as the starting locations for mesh smoothing:

*ADAPTIVE MESH CONTROLS, NAME=*name*,
MESHING PREDICTOR=CURRENT

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Meshing predictor: Current deformed position**

Positioning Nodes in Eulerian Domains

Mesh sweeps can be based on the locations of nodes at the end of the previous adaptive mesh increment. This technique is recommended for problems that are Eulerian in nature, where material flow is significant compared to overall deformation. Therefore, it is the default for adaptive mesh domains with one or more Eulerian boundary regions. This approach will result in a virtually stationary mesh.

Input File Usage: Use the following option to use the position of the nodes at the end of the previous adaptive mesh increment as a starting location for mesh smoothing:

*ADAPTIVE MESH CONTROLS, NAME=*name*,
MESHING PREDICTOR=PREVIOUS

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Meshing predictor: Position from previous adaptive mesh increment**

Solution-Dependent Meshing Based on Concave Boundary Curvature

Mesh smoothing algorithms based only on minimizing element distortion tend to reduce the mesh refinement in areas of concave curvature, especially as the curvature evolves. Having sufficient mesh refinement near highly curved boundaries is often important to model both the shape and volume of the domain. To prevent the natural reduction in mesh refinement of areas near evolving concave curvature, Abaqus/Explicit uses solution-dependent meshing to focus mesh gradation toward these areas automatically.

Although solution-dependent meshing may "pull" more elements into areas of high curvature, its primary purpose is to retain the nominal refinement in these zones. Therefore, a fine mesh should always be used when and where highly curved boundaries are expected and solution-dependent meshing should generally not be used as a direct substitute for more elements.

The aggressiveness of the solution-dependent meshing is governed by the curvature refinement weight, α_C. By default, $\alpha_C = 1$, which corresponds to an aggressivity level that has been chosen to work well on a wide variety of problems. You can change the curvature refinement weight. A value of zero indicates no solution dependence due to evolving boundary curvature, and a value greater than one increases the aggressivity from the default.

Input File Usage: *ADAPTIVE MESH CONTROLS, NAME=*name*,
CURVATURE REFINEMENT=α_C

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Curvature refinement: α_C**

Smoothing a Distorted Mesh at the Beginning of a Step

When an adaptive mesh domain contains a structured mesh of uniform density, the mesh will move independently from the material only when the domain deforms. If the mesh is initially nonuniform, the meshing algorithms in Abaqus/Explicit will smooth the mesh even in the absence of deformation or material transport.

When the initial mesh contains highly distorted elements, it is often useful to smooth the mesh before the step begins so that the best possible mesh is used throughout the step. When a uniform smoothing objective is used, five mesh sweeps are performed by default at the beginning of the step in which the adaptive mesh domain is defined. For a graded smoothing objective, two mesh sweeps are performed by default at the beginning of the step without accounting for gradation. The aspect ratios used for gradation in all subsequent mesh sweeps are based on this locally smoothed mesh.

Initial conditions are advected to the new mesh when initial mesh sweeps are performed.

Input File Usage: Use the following option to change the number of mesh sweeps that will be performed at the beginning of the first step in which the adaptive mesh definition is active:

**ADAPTIVE MESH*, INITIAL MESH SWEEPS=*number of initial sweeps*

For example, the following option would smooth a badly distorted mesh with 15 mesh sweeps at the beginning of the step, before performing adaptive meshing with three mesh sweeps every 20 increments throughout the step:

**ADAPTIVE MESH*, FREQUENCY=20, MESH SWEEPS=3,
INITIAL MESH SWEEPS=15

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Initial remeshing sweeps: Value:** *number of initial sweeps*

Meshing on Boundary Regions

Adaptive meshing on Lagrangian and sliding boundary regions is subject to the constraint that the mesh and material must move together in the direction normal to the boundary. Nodes on the interior of such a boundary are allowed to slide freely over the material within the boundary region, which maximizes the amount of mesh smoothing that can be performed. Nodes are positioned in each mesh sweep by applying the basic smoothing methods while constraining the nodes to lie on the boundary region. In three dimensions some nodes on Lagrangian boundary regions will be on a Lagrangian edge. In each mesh sweep these nodes are positioned by applying the basic smoothing methods while constraining the node to lie along the discrete Lagrangian edge.

For problems in which the flow of material from element to element along the boundary is significant compared to the deformation, oscillations in the boundary mesh can result if these constraints are applied symmetrically with respect to the upstream and downstream directions of material flow. Abaqus/Explicit uses a Petrov-Galerkin weighting of the free boundary constraint to suppress any oscillations. The algorithm is volume preserving, and the degree of upwinding is chosen automatically.

Advecting Solution Variables to the New Mesh

The framework for adaptive meshing in Abaqus/Explicit is the Arbitrary Lagrangian-Eulerian method, which introduces advective terms into the momentum balance and mass conservation equations to account for independent mesh and material motion. There are two basic ways to solve these modified equations: solve the nonsymmetric system of equations directly, or decouple the Lagrangian (material) motion from the additional mesh motion using an operator split. The operator split method is used in Abaqus/Explicit because of its computational efficiency. Furthermore, this technique is appropriate in an explicit setting because small time increments limit the amount of motion within a single increment.

In an adaptive meshing increment the element formulations, boundary conditions, external loads, contact conditions, etc. are handled first in a manner consistent with a pure Lagrangian analysis. Once the Lagrangian motion is updated and mesh sweeps have been performed to find the new mesh, the solution variables are

remapped by performing an advection sweep. The advection sweep accounts for the advective terms in the momentum balance and continuity equations.

Advection Methods for Element Variables

Element and material state variables must be transferred from the old mesh to the new mesh in each advection sweep. The number of variables to be advected depends on the material model and element formulation; however, stress, history variables, density, and internal energy are always solution variables. Two methods are available for the advection of element variables: the default second-order method based on the work of Van Leer (Van Leer, 1977) and a first-order method based on donor cell differencing.

Both advection methods incorporate the concept of upwinding. They also conserve the element variables in an integral sense when mapping from the old mesh to the new mesh (that is, the value of any solution variable integrated over the domain is unchanged by adaptive meshing). Using a conservative algorithm to advect the element density and the internal energy automatically ensures conservation of mass and energy for an adaptive mesh domain without Eulerian boundary regions.

Both advection methods are also monotonic and consistent. A method is monotonic if an element quantity with a monotonic, increasing spatial distribution over a portion of the old mesh remains as such in the new mesh. A method is consistent if, when solution variables are advected to a new mesh that is identical to the old mesh, all element quantities remain unchanged.

Second-Order Advection

Second-order advection is used by default for all adaptive mesh domains. It is recommended for all problems, ranging from quasi-static to transient dynamic shock. An element variable, ϕ, is remapped from the old mesh to the new mesh by first determining a linear distribution of the variable in each old element, as illustrated in *Figure 2* for a simple one-dimensional mesh.

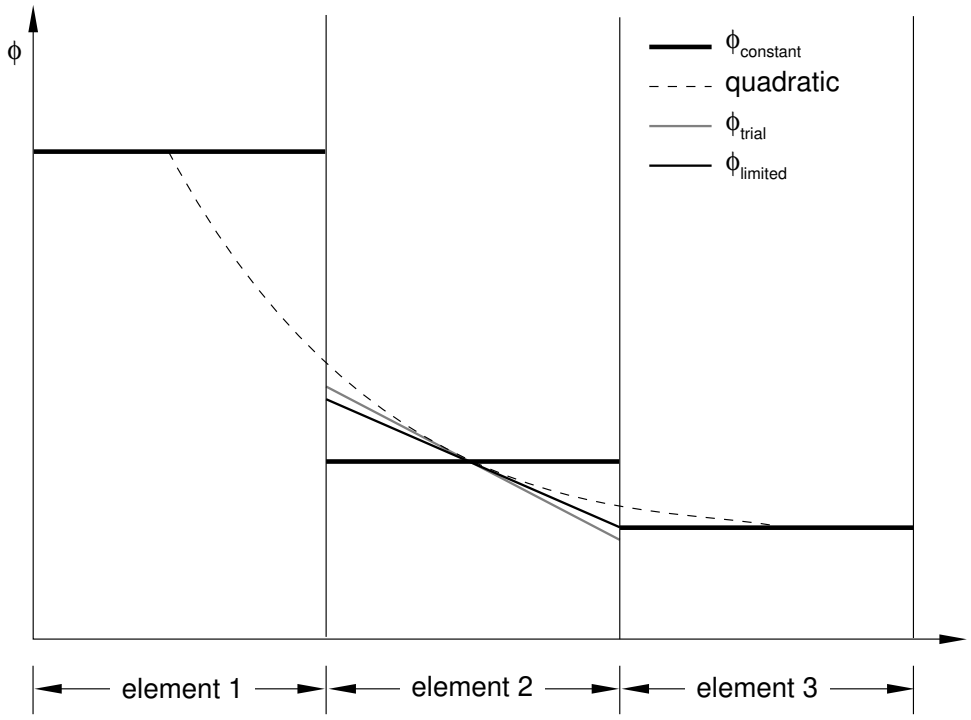


Figure 2: Second-order advection.

The linear distribution of ϕ in the middle element depends on the values of ϕ in the two adjacent elements. To construct the linear distribution:

1. A quadratic interpolation is constructed from the constant values of ϕ at the integration points of the middle element and in its adjacent elements.
2. A trial linear distribution, ϕ_{trial}, is found by differentiating the quadratic function to find the slope at the integration point of the middle element.
3. The trial linear distribution in the middle element is limited by reducing its slope until its minimum and maximum values are within the range of the original constant values in the adjacent elements. This process is referred to as flux-limiting and is essential to ensure that the advection is monotonic.

Once the flux-limited linear distributions are determined for all elements in the old mesh, these distributions are integrated over each new element. The new value of the variable is found by dividing the value of each integral by the new element volume. (See *Figure 3* for a first-order example of this calculation.)

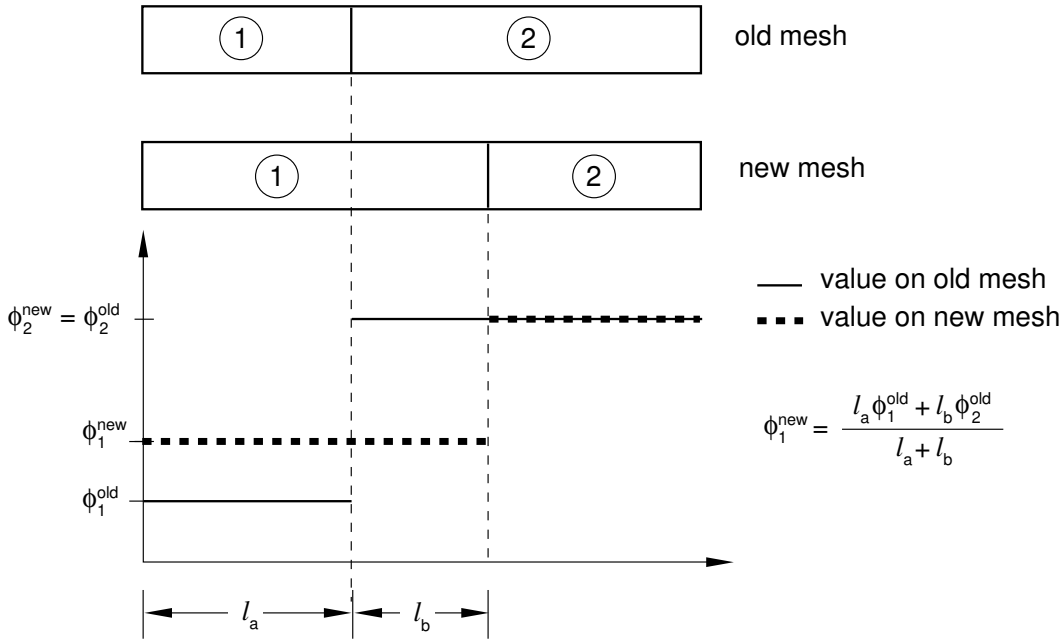


Figure 3: First-order advection.

Input File Usage: Use the following option to specify that the second-order advection method should be used to remap element variables:

**ADAPTIVE MESH CONTROLS*, NAME=*name*, ADVECTION=SECOND ORDER

Abaqus/CAE Usage: Step module: **Other**->**ALE Adaptive Mesh Controls**->**Create**: **Name**: *name*, toggle on **Second order**

First-Order Advection

First-order advection is simple and computationally efficient; however, it tends to diffuse sharp gradients over time, especially in transient dynamic analyses or other problems that require fairly frequent adaptive meshing. Therefore, this technique should be used only as a computationally efficient alternative for quasi-static simulations that do not require frequent adaptive meshing.

Figure 3 illustrates the first-order method for a portion of a one-dimensional mesh. An element variable, ϕ, is remapped from the old mesh to the new mesh by first assuming a constant value of the variable for each old element. These values are then integrated over each new element. The new value of the variable is found by dividing the value of each integral by the new element volume.

Input File Usage: Use the following option to specify that the first-order advection method should be used to remap element variables:

**ADAPTIVE MESH CONTROLS*, NAME=*name*, ADVECTION=FIRST ORDER

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, toggle on **First order**

Momentum Advection

Nodal velocities are computed on a new mesh by first advecting momentum, then using the mass distribution on the new mesh to calculate the velocity field. Advecting momentum directly ensures that momentum is conserved properly in the adaptive mesh domain during remapping. Two methods are available for advecting momentum: the default element center projection method and the half-index shift method (Benson, 1992). Both methods are applicable for all adaptive mesh applications.

Element Center Projection Method

The element center projection method is the default method used to advect momentum and requires the fewest numerical operations. The element momentum is calculated first for the old mesh based on the mass and velocity of the element nodes. The element momentum is then advected from the old mesh to the new mesh by the same first- or second-order algorithms used for advecting element variables. Finally, the element momentum on the new mesh is assembled at the nodes using a projection. The element center projection method requires the advection of only two or three extra variables in two dimensions or three dimensions, respectively.

Input File Usage: Use the following option to request the most computationally efficient momentum advection method:

**ADAPTIVE MESH CONTROLS*, NAME=*name*,
MOMENTUM ADVECTION=ELEMENT CENTER PROJECTION

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Momentum advection: Element center projection**

Half-Index Shift Method

The half-index shift method is computationally more intensive than the element center projection method, but it might result in less wave dispersion for some problems. This method first shifts each of the nodal momentum variables from the nodes surrounding an element to the element center. The shifted momentum variables are then advected from the old mesh to the new mesh by the same first- or second-order algorithms used for advecting element variables, providing momentum variables at the center of the new elements. Finally, the momentum variables at the element centers in the new mesh are shifted back to the nodes. The half-index shift method requires the advection of 8 or 24 extra variables in two or three dimensions, respectively, which can increase the cost of each advection sweep significantly.

Input File Usage: Use the following option to specify that the half-index shift method should be used for momentum advection:

ADAPTIVE MESH CONTROLS, NAME=*name*,
MOMENTUM ADVECTION=HALF INDEX SHIFT

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Momentum advection: Half-index shift**

References

- Benson, D. J., "Momentum Advection on a Staggered Mesh," *Journal of Computational Physics*, vol. 100, pp. 143–162, 1992.
- Van Leer, B., "Towards the Ultimate Conservative Difference Scheme III. Upstream-centered Finite-Difference Schemes for Ideal Compressible Flow," *Journal of Computational Physics*, vol. 23, pp. 263–275, 1977.
- Van Leer, B., "Towards the Ultimate Conservative Difference Scheme IV. A New Approach to Numerical Convection," *Journal of Computational Physics*, vol. 23, pp. 276–299, 1977.

Modeling Techniques for Eulerian Adaptive Mesh Domains in Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References:

- *About ALE Adaptive Meshing*
- *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*
- **ADAPTIVE MESH CONSTRAINT*
- *Understanding ALE adaptive meshing*

Overview

An Eulerian adaptive mesh domain:

- is used to model material flowing through the mesh; and
- typically has two Eulerian boundary regions, one inflow and one outflow, connected by Lagrangian and/or sliding boundary regions.

The correct combination of mesh constraints and material boundary conditions applied to an Eulerian boundary region depends on whether the region acts as an inflow or an outflow boundary. The region types and mesh constraints assigned to the boundary regions that are connected to the Eulerian boundary regions must be chosen to simulate the correct physical behavior as well.

The adaptive meshing technique in Abaqus/Explicit is robust if the mesh is underconstrained: the modeling techniques presented in this section are intended to provide guidance in properly defining Eulerian models.

ALE Adaptive Mesh Constraints on Eulerian Boundaries

ALE adaptive mesh constraints should be applied normal to an Eulerian boundary region; otherwise, the motion of the mesh on the boundary is ambiguous. If no mesh constraints are applied normal to the boundary, Abaqus/Explicit will treat the region as if it were sliding, and the mesh will follow the material normal to the boundary.

Although there are no restrictions on specifying adaptive mesh constraints at nodes on an Eulerian boundary region, the following guidelines should be followed in most cases:

- Mesh constraints should be applied to every node on the Eulerian boundary region, including the corners and edges.
- Mesh constraints should be applied either only normal to the Eulerian boundary region or in every direction. Mesh constraints should not be specified in only the direction tangential to an Eulerian boundary region; such constraints are ambiguous and may result in undesirable motion of the mesh at the boundary.

Loads and boundary conditions on Eulerian boundaries act on the material that instantaneously coincides with the mesh at the surface. When used in combination with spatial adaptive mesh constraints, physically meaningful Eulerian flow conditions can be defined.

Defining Inflow Eulerian Boundaries

The material flowing into an adaptive mesh domain through an Eulerian boundary will have the same stress and material state as the material in the elements immediately adjacent to the boundary. Therefore, it is important to

maintain the stress and material state in those elements at the desired values (which in many cases will be zero, to simulate stress-free material entering the Eulerian domain). To accomplish this goal:

- position the inflow boundary far enough upstream from high solution gradients to ensure that the response at the inflow boundary is smooth, and
- apply sufficient mesh and material constraints at the boundary (as described later in this section).

To be physically meaningful, the size and shape of the inflow boundary region must be maintained. For example, applying sufficient constraints is crucial for steady-state process simulations where the cross-section of the workpiece entering the adaptive mesh domain is known and affects the response downstream. The types of constraints appropriate for an inflow boundary depend on whether the precise location of the inflow boundary region is known or whether it is part of the solution.

Known Inflow Boundary Location

In many problems the area, shape, and position of the inflow boundary are known a priori. For example, in the steady-state analysis of a forward extrusion process, an inflow Eulerian boundary can be used to model the flow of material into the adaptive mesh domain. The size of the inflow boundary is based on the known billet cross-section, and the location of the inflow boundary is fixed because of the confined conditions on the material.

When the area, shape, and location of the inflow boundary are known, both material and mesh constraints should be applied. *Figure 1* shows a typical model setup for a two-dimensional forward extrusion problem where either a prescribed mass flow rate or a prescribed uniform pressure is applied to a known inflow boundary. Apply boundary conditions at all nodes on the inflow boundary region to prescribe material constraints in the directions tangential to the boundary surface. Preventing motion of the material tangential to the inflow boundary helps to maintain the stress and material state of the elements adjacent to the Eulerian boundary.

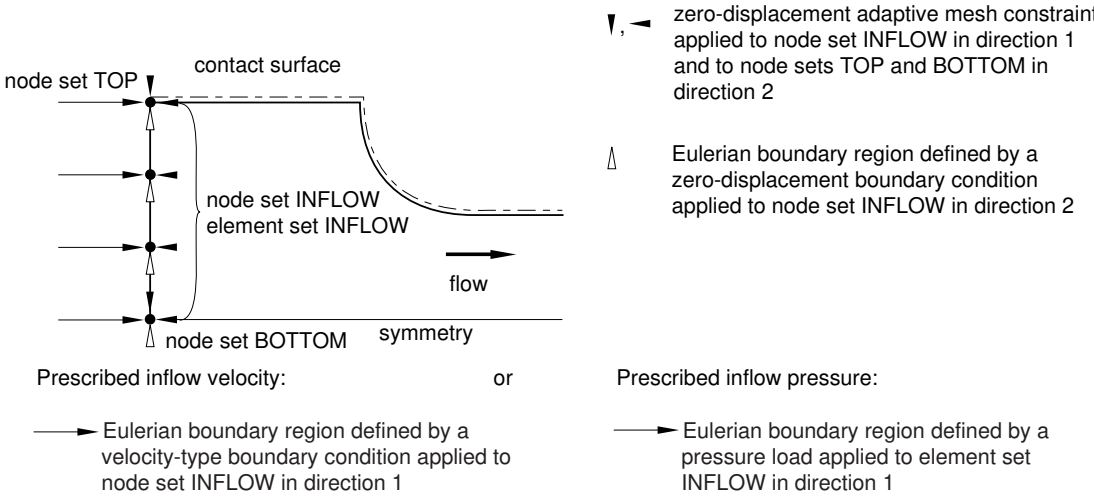


Figure 1: Known inflow boundary.

Apply adaptive mesh constraints in the normal direction at all nodes on the inflow boundary. In addition, apply mesh constraints in all tangential directions at the edges and corners surrounding the Eulerian boundary region. These constraints fix the location and size of the cross-sectional area at the inflow boundary.

If a nonuniform boundary condition or load is applied to the material at the inflow boundary or if the initial material state in the elements adjacent to the boundary is nonuniform in the tangential direction, apply tangential mesh constraints to the nodes strictly in the interior of the Eulerian boundary region.

Although the application of mesh and material constraints tangential to and along the edges and corners of an inflow Eulerian boundary may appear to be redundant, they are actually independent. For example, consider a long billet with a variable cross-section, as shown in *Figure 2*.

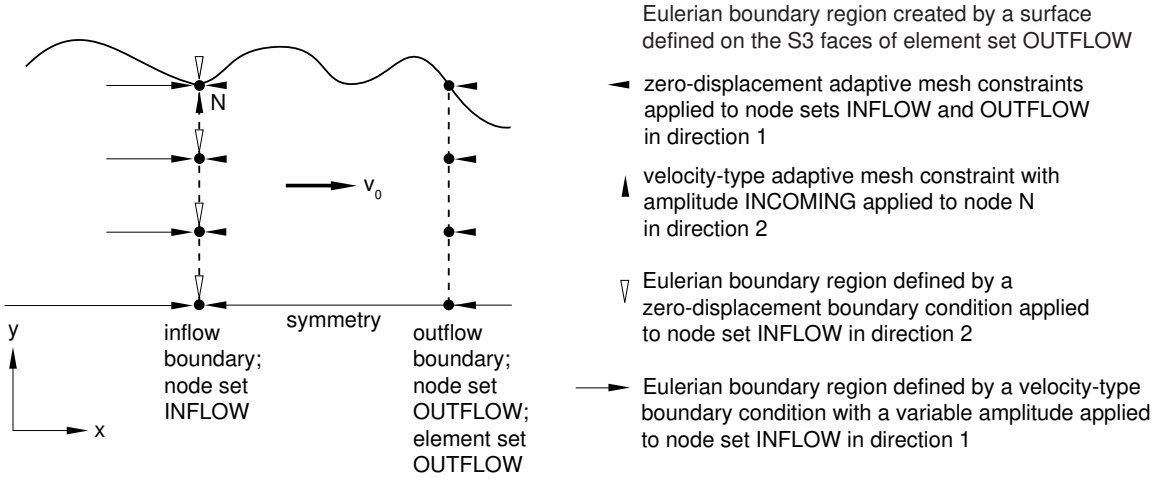


Figure 2: Modeling a billet with a variable cross-section.

The adaptive mesh domain, with its inflow and outflow Eulerian boundary regions, is assumed to represent a portion of the billet along its length. The entire billet moves with a rigid body velocity along its length (x-direction) so that material flows into one Eulerian boundary and out the other. Boundary conditions are applied to the material at the inflow boundary to maintain this velocity. Mesh constraints are applied normal to the inflow and outflow boundary regions. The mesh constraint applied in the y-direction at node N is used to prescribe the known variable incoming cross-section of the material. The motion of this node does not affect the velocity field of the material entering the domain.

Unknown Inflow Boundary Location

Sometimes, the location of the inflow boundary region is known only approximately; its precise location will be determined from the solution. For these problems, apply adaptive mesh constraints only in the direction normal to the Eulerian boundary region. In the absence of tangential mesh constraints at the edges and corners of the Eulerian boundary region, Abaqus/Explicit will move these edges and corners with the material in the tangential direction but with the mesh constraints in the normal direction. Therefore, material constraints should be applied using multi-point constraints (see *General Multi-Point Constraints*) or linear constraint equations (see *Linear Constraint Equations*) to preserve the cross-sectional area of the inflow boundary.

For example, consider a steady-state rolling simulation with multiple rollers in an asymmetric configuration, as shown in *Figure 3*.

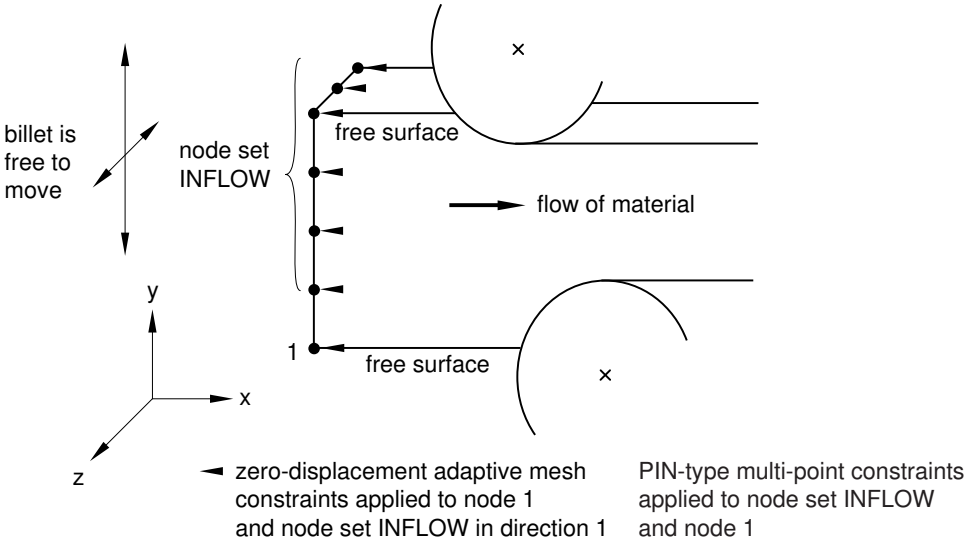


Figure 3: Unknown inflow boundary location.

It may be impractical to extend the analysis domain as far as the guides on the upstream side, but spatially fixing the inflow boundary at an arbitrary position in the *y*- and *z*-directions may cause unrealistic stress on the workpiece as it finds an equilibrium position between the rollers. Mesh constraints are applied normal to the Eulerian boundary region to fix the position of the inflow boundary relative to the rollers in the *x*-direction. Material constraints (applied with a PIN MPC) are used to ensure that material enters the domain at a uniform velocity and that the cross-section does not rotate. The material constraints will maintain the cross-sectional shape of the section while allowing it to move laterally to the correct equilibrium position. Since tangential mesh constraints are not used, the mesh will follow the material in the directions tangential to the Eulerian boundary region.

Defining Outflow Eulerian Boundaries

Typically, adaptive mesh constraints should be applied only in the direction normal to the surface on an Eulerian boundary region that acts as an outflow boundary. No tangential mesh constraints should be applied to the edges or corners of an outflow boundary adjacent to a Lagrangian (or sliding) boundary region acting as a free surface. In contrast to inflow boundaries, the cross-section of an outflow boundary adjacent to a free surface is determined by the solution in the domain. At the edge or corner where an Eulerian boundary region meets a Lagrangian or sliding boundary region, Abaqus/Explicit will satisfy the applied mesh constraint normal to the Eulerian boundary region and the inherent mesh constraint normal to the Lagrangian or sliding boundary region simultaneously, thus correctly handling the evolution of the free surface adjacent to the outflow boundary. *Figure 4* shows the evolution of an outflow boundary from t_0 to t_1, where material continues to flow through the outflow boundary.

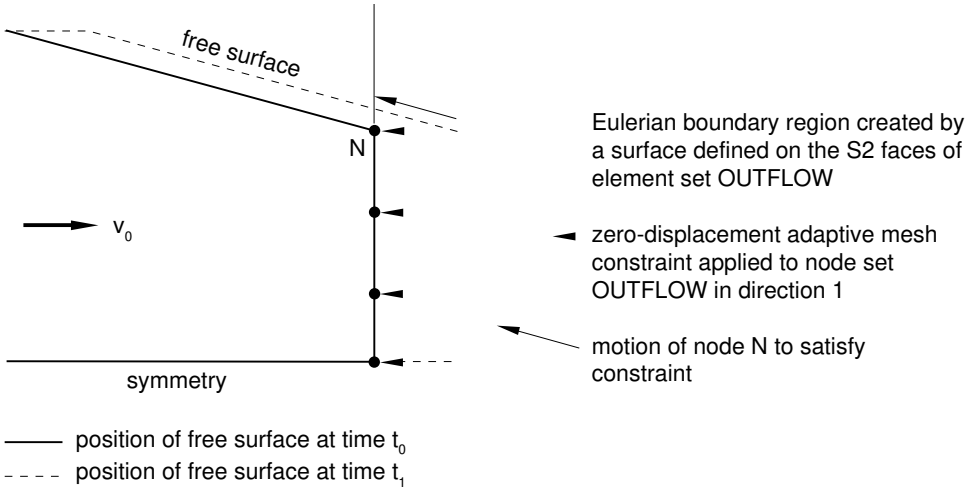


Figure 4: Abaqus/Explicit will respect the free surface at an Eulerian outflow boundary.

The mesh constraint normal to the Eulerian outflow boundary is applied by moving node N along the free surface of the material, so that the outflow boundary "expands" with the material arriving from upstream. Although not shown in the figure, mesh smoothing causes all other nodes on the outflow boundary, with the exception of the node on the symmetry plane, to move up toward node N as the boundary expands.

No special material boundary conditions are required at outflow Eulerian boundaries. Boundary conditions tangential to the outflow boundary are recommended only if they are the same as those defined upstream (e.g., a symmetry plane running along the length of an Eulerian domain). However, to improve convergence to the steady-state solution in steady-state process simulations, it is often useful to constrain the material velocity to be uniform normal to the outflow boundary using multi-point constraints or linear constraint equations.

Defining Eulerian Boundary Regions That Act as Both Inflow and Outflow Boundaries

Although it is rarely appropriate, an Eulerian boundary region can act as both an inflow and an outflow boundary at different times during the same analysis step. Adaptive mesh constraints and material boundary conditions at such a boundary should be chosen to be physically meaningful for both inflow and outflow situations.

For each node on the edges and corners of an Eulerian boundary region that does not have mesh constraints tangential to the boundary surface, Abaqus/Explicit will determine in each adaptive mesh increment whether the boundary at the node is acting as an inflow or an outflow boundary. If an inflow condition is detected, the node will move with the material in the tangential direction but with the mesh constraints in the normal direction. If an outflow condition is detected, the movement of the node will both follow the adjacent Lagrangian boundary region and satisfy the mesh constraint normal to the Eulerian boundary region.

Lagrangian Versus Sliding Boundary Regions on Eulerian Domains

Many applications using Eulerian adaptive mesh domains, including the simulation of steady-state processes, have a primary direction of material flow and use a control volume approach to model the process zone. These problems usually include two Eulerian boundary regions, representing an inflow boundary and outflow boundary. The remaining surfaces between the Eulerian boundaries can be either Lagrangian or sliding boundary regions. Determining which type of boundary region to use between the two Eulerian boundary regions depends on the type of load or boundary condition that is required:

- Use a sliding boundary region to define boundary conditions or loads that act at a spatial location on a portion of the surface along the length of the control volume. Apply adaptive mesh constraints to fix the mesh spatially in the flow direction (and possibly in the direction transverse to the flow). For example, a distributed pressure can be applied around the circumference of the control volume, as shown in *Figure 5*.

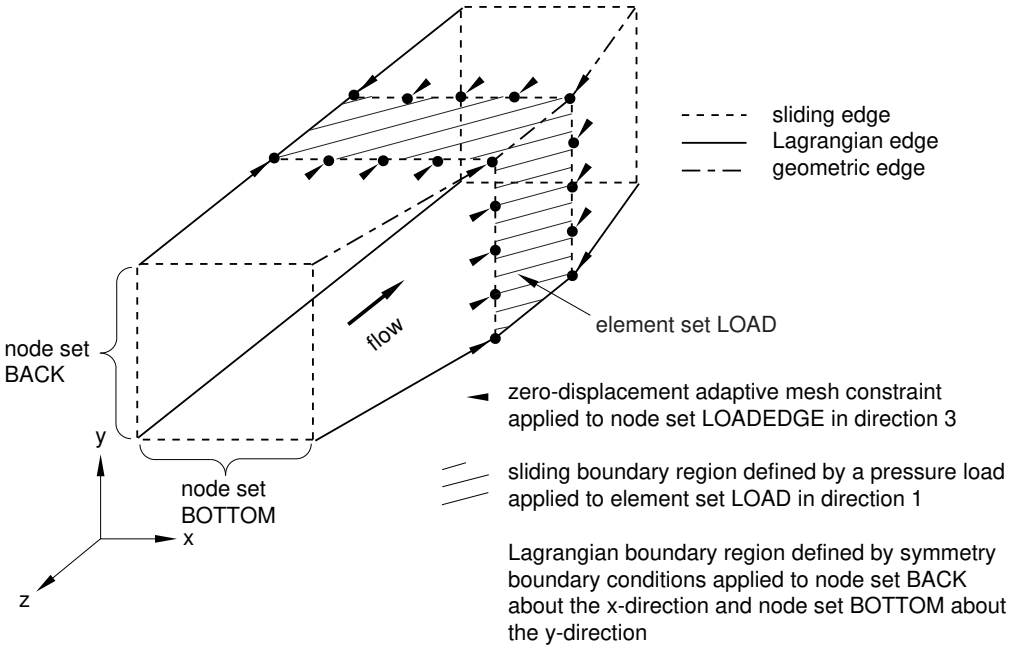


Figure 5: Applying a spatial pressure load to a portion of the surface along the length of an Eulerian control volume.

The distributed pressure load is defined using a sliding boundary region. Mesh constraints are applied to fix the boundary region spatially in the flow direction. Similarly, a concentrated load could be applied to a specific spatial location to model the effect of a very sharp body pressing into the workpiece at a known location with a known force.

- Use a sliding boundary region to define boundary conditions or loads that act along the entire length of the Eulerian control volume between the inflow and outflow boundaries and act in a spatial manner transversely to the flow. If the load acts on only a portion of the surface in the transverse direction, it may be necessary to apply mesh constraints in the direction transverse to the flow. For example, a boundary condition that acts as a knife edge along the length of the domain is shown in *Figure 6*. Mesh constraints are applied in the transverse direction (and, if the line of application is curved, along the line) to keep the boundary condition fixed spatially.

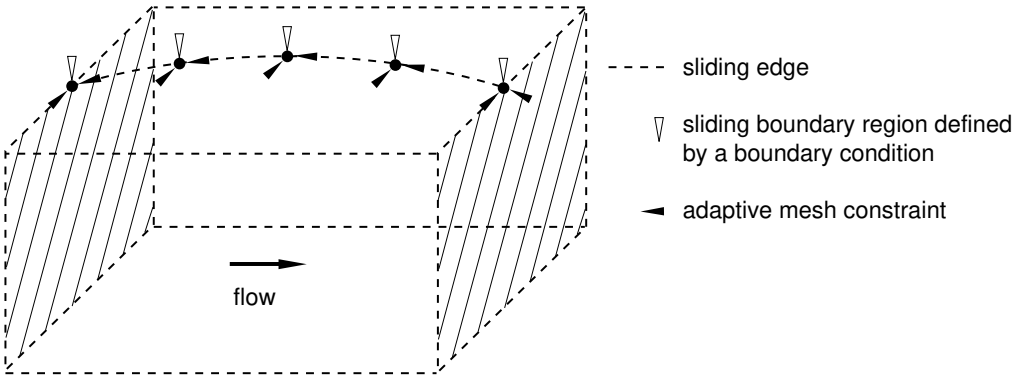


Figure 6: Applying a boundary condition along the entire length of the Eulerian control volume.

- Use a Lagrangian boundary region (default) to define boundary conditions or loads that act along the entire length of the surface of the Eulerian control volume between the inflow and outflow boundary and act in a Lagrangian manner transversely to the flow. In three dimensions, symmetry conditions should typically act

in a Lagrangian manner transverse to the flow direction. In many cases geometric edges will prevent material from flowing off the symmetry plane and onto the free surface. However, since geometric edges can be deactivated as surfaces flatten, Lagrangian boundary regions should be used to define the symmetry planes for these types of problems. In *Figure 5* quarter-symmetry is assumed, and the symmetry planes are defined using Lagrangian boundary regions. The resulting Lagrangian edges that run from one Eulerian boundary to the other separate the symmetry planes from the free surface.

- Boundary conditions or loads that act on only a specific portion of the material between the inflow and outflow boundaries cannot usually be modeled for problems utilizing Eulerian control volumes. Since the mesh underneath the load or boundary condition must follow the material, it will eventually be restricted by the Eulerian boundary. This treatment of loads and boundary conditions is not usually consistent with a steady-state model and should not arise in practical simulations using Eulerian adaptive mesh domains.

Output and Diagnostics for ALE Adaptive Meshing in Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Output to the Output Database*
- *About ALE Adaptive Meshing*
- *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*
- **ADAPTIVE MESH*
- **ADAPTIVE MESH CONTROLS*
- **DIAGNOSTICS*
- **TRACER PARTICLE*
- *Using display groups to display subsets of your model*

Overview

Output for ALE adaptive meshing:

- can be used to verify the automatic splitting of user-defined domains, the formation of Lagrangian edges and corners, the formation of geometric edges and corners, and the determination of nonadaptive nodes;
- must be interpreted carefully, since the values of output variables at specific locations in the mesh are no longer linked to values at particular material points;
- can include the definition of tracer particles, which follow material points and allow you to examine the trajectory of those points and plot material time histories of all element and nodal variables at those points; and
- can include diagnostic information on the efficiency of adaptive meshing and the accuracy of advection.

Verifying the Model

Output that can be used to verify adaptive meshing models is available in the data (`.dat`) file and in the output database (`.odb`) (see *About Output* for details on these files).

Element Sets

When user-defined adaptive mesh domains are split by Abaqus/Explicit, the elements that compose the new subdivided domains are printed to the data (`.dat`) file.

New element sets are created and written to the output database (`.odb`) for all adaptive mesh domains. The name of the element set created for each domain is the user-defined name, plus the number of the subdivision (1 if no subdivisions were created), plus the step number. For example, if the user-defined adaptive mesh domain specified for the element set *domain_name* spanned three disjoint parts, Abaqus/Explicit would subdivide the user-defined domain into three domains and create three element sets in the output database (`.odb`) for the first step: *domain_name*`-1-1`, *domain_name*`-2-1`, and *domain_name*`-3-1`.

Abaqus/CAE can be used to verify the creation of the subdivided domains.

Edges and Nonadaptive Nodes

Abaqus/Explicit automatically forms Lagrangian edges and corners and identifies nonadaptive nodes based on the topology of the adaptive mesh domains, connections to nonadaptive domains, and user-specified boundary

regions. Furthermore, geometric edges and corners are formed automatically based on the initial geometry and the value of the initial feature angle. See *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit*. Lagrangian edges, geometric edges and corners, and nonadaptive nodes (including Lagrangian corners) are output to the data (`.dat`) file for each adaptive mesh domain. This information can be obtained by requesting a history definition summary printout to the data file (see *Model and History Definition Summaries*) or by monitoring the progress of the adaptive meshing (see *Monitoring the Progress of ALE Adaptive Meshing* below).

In addition, up to three node sets are created in the output database (`.odb`) for each adaptive mesh domain in each step. The names of the node sets are created by concatenating the following information:

- the domain element set name;
- the number of the subdivision (1 if no subdivisions were created);
- the letters LE for Lagrangian edge, GE for geometric edge or corner, or NA for nonadaptive nodes (including Lagrangian corners); and
- the step number.

For example, if a user-defined three-dimensional adaptive mesh domain specified for element set *domain_name* is subdivided automatically into two adaptive mesh domains, Abaqus/Explicit will generate up to six node sets in the output database for the first step: *domain_name*`-1-LE-1`, *domain_name*`-1-GE-1`, *domain_name*`-1-NA-1`, *domain_name*`-2-LE-1`, *domain_name*`-2-GE-1`, and *domain_name*`-2-NA-1`.

Since boundary regions are separated by corners, not edges, in two dimensions, node sets will not be created for Lagrangian edges in two-dimensional adaptive mesh domains. The Lagrangian corners are included in the nonadaptive (NA) node set, as for three-dimensional domains.

Abaqus/CAE can be used to verify the creation of Lagrangian edges and corners, geometric edges and corners, and nonadaptive nodes.

Interpreting Results

When adaptive meshing is not performed, the finite element mesh follows the material, which enables a straightforward interpretation of analysis results. You can visualize deformation and material motion by studying the motion of the mesh. Each nodal and element output variable corresponds to a specific material location, because the mesh is fixed to the same material point throughout time.

Once adaptive meshing takes place, the locations of mesh and material points deviate, and analysis results must be interpreted accordingly. The motion of the mesh on the interior of an adaptive mesh domain represents the composite effects of the material motion and adaptive meshing. The motion of the mesh and the motion of the material on Lagrangian and sliding boundary regions is identical in the direction normal to the boundary but not in the direction tangential to it.

Nodal Variables

When adaptive meshing is performed, a material point that is coincident with a node at the beginning of the step may not remain coincident with that node throughout the step. Values of displacement and current coordinates represent the motion of the node, not necessarily the motion of the material. All other nodal variables—including velocity, acceleration, and reaction forces—represent the value of the variable for the material particle at the current location of the node. Contour or vector plots of these variables will show their correct spatial distribution and are, therefore, meaningful. However, time histories of nodal variables for nodes that undergo adaptive meshing are generally not meaningful. In steady-state problems, though, a velocity or acceleration time history based at a fixed spatial location rather than at a specific material point may be useful.

Element Variables

Similarly, when adaptive meshing is performed, a material particle that is coincident with an element integration point at the beginning of a step may not remain so throughout the step. Therefore, element integration point

variables do not necessarily represent values at the same material point throughout the step. Contour or vector plots of element integration point variables are meaningful for the same reasons described for nodal variables. However, time histories are based at the spatial location of the element integration point and not at a specific material point.

Whole element variables have a similar interpretation.

Tracking Nodal or Element Variables at Material Points

Tracer particles can be defined to track material points in an adaptive mesh domain. These particles can also be used to obtain time histories of nodal or element integration point variables that correspond to the time variation of the variable at a specific material point. Tracer particles are defined as described below (see *Output to the Output Database* for more information). Node and element variable output can be requested for tracer particle sets to examine the trajectory of material particles or to obtain material time histories. Output for tracer particles can be written only to the output database (`.odb`).

Using Tracer Particles in Lagrangian Domains

In most adaptive meshing simulations using Lagrangian domains, the nodes and elements in the domain correspond neither to a specific spatial location nor to a specific material point or volume. Thus, time histories of variables at nodes and at element integration points are often physically meaningless in a Lagrangian adaptive mesh domain. Tracer particles should be defined to view time history information. Tracer particles can also be used to visualize the motion of the material.

The initial location of a tracer particle is defined to be coincident with a node, termed the parent node. Tracer particles are defined in sets by defining multiple parent nodes or node sets. You indicate the nodes whose current locations correspond to the initial location of the tracer particles and assign a name to the tracer particle set to identify it for use in output requests. Tracer particles are released from their parent nodes repeatedly at specified intervals during the step in which they are defined. The particles follow material points for the remainder of that step and in all subsequent steps.

Tracer particles are typically defined only on adaptive mesh domains, although they can be defined on nodes connected to any low-order solid element in the model. For analyses in which adaptive meshing is not performed until later steps, tracer particles can be defined on nonadaptive domains at the beginning of an analysis and will be tracked continuously as the domain becomes adaptive. Similarly, tracer particles will be tracked from domain to domain if adaptive mesh domain topologies change from step to step.

Input File Usage: Use the following option to define a tracer particle set:

**TRACER PARTICLE*, TRACER SET=*tracer_set_name*
list of tracer particle parent nodes

Abaqus/CAE Usage: Tracer particles are not supported in Abaqus/CAE.

Using Tracer Particles in Eulerian Domains

Time histories at nodes and element integration points in an Eulerian domain may have physical meaning at points where spatial adaptive mesh constraints are applied. For example, the time variation of equivalent plastic strain in elements along an outflow Eulerian boundary acts as a spatial time history of that variable and can be used to evaluate whether the process has reached a steady-state solution.

Tracer particles can be defined to evaluate the material time history of variables at a material point as it flows through the Eulerian domain. Tracer particles can also be used to evaluate the trajectory and path of material points as they pass through the domain.

Tracer particles can be assigned to any parent node in an Eulerian adaptive mesh domain. If a tracer particle reaches an outflow boundary and material continues to flow out, the tracer particle will no longer be tracked and all output history variables associated with the tracer particle will be zero after deactivation.

When material flow through the mesh domain is significant, sets of tracer particles can be released from the current locations of the parent nodes at multiple times during the step. Each release of tracer particles is referred to as particle birth. After particle birth the tracer particles follow the motion of the material regardless of the motion of the mesh. You can indicate the number of particle birth stages in a step. These stages will be evenly spaced throughout the time period of the step.

For example, a tracer particle set can be defined such that all nodes along an inflow Eulerian boundary are parent nodes. Multiple birth stages can be specified so that a set of tracer particles is released from the mesh at the inflow boundary periodically during the step. If enough birth stages are defined, the domain will eventually be spanned with tracer particles as material flows from the inflow boundary to the outflow boundary.

Input File Usage: Use the following option to define a tracer particle set with multiple birth stages:

**TRACER PARTICLE*, TRACER SET=*tracer_set_name*, PARTICLE BIRTH STAGES=*n*
list of tracer particle parent nodes

Abaqus/CAE Usage: Tracer particles are not supported in Abaqus/CAE.

Monitoring the Progress of ALE Adaptive Meshing

Diagnostic information can be written to the message (`.msg`) file to track the efficiency and accuracy of adaptive meshing. You can select the level of diagnostic output that is written.

Obtaining a Summary at the End of a Step

By default, a summary of adaptive meshing information for each adaptive mesh domain will be written to the message (`.msg`) file at the end of each step. This summary information includes:

- the average percentage of nodes moved,
- the maximum percentage of nodes moved,
- the minimum percentage of nodes moved, and
- the average number of advection sweeps.

Each value is calculated for a single adaptive mesh domain over all adaptive mesh increments. The cost of advection is approximately proportional to the percentage of nodes moved, since variables are not advected for elements that have not been relocated during adaptive meshing.

Input File Usage: Use the following option to request a summary for each adaptive mesh domain at the end of each step:

**DIAGNOSTICS*, ADAPTIVE MESH=STEP SUMMARY

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Obtaining a Summary for Every ALE Adaptive Mesh Increment

In addition to the step summary information, the following diagnostics can be obtained for each adaptive mesh domain at every adaptive mesh increment:

- the percentage of nodes moved, and
- the number of advection sweeps.

Input File Usage: Use the following option to obtain summary information at the end of the step and at every adaptive mesh increment:

**DIAGNOSTICS*, ADAPTIVE MESH=SUMMARY

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Obtaining Details of Advection Accuracy for Every ALE Adaptive Mesh Increment

The following detailed diagnostic information can also be written to the message (`.msg`) file to track the accuracy of the advection:

- mass and momentum before and after advection, and
- percentage volume change.

Input File Usage: Use the following option to request the most detailed diagnostics, which include advection accuracy measures and summary information for each adaptive mesh domain, reported at every adaptive mesh increment:

**DIAGNOSTICS*, ADAPTIVE MESH=DETAIL

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Suppressing ALE Adaptive Mesh Diagnostics

You can suppress output of all adaptive mesh diagnostic information.

Input File Usage: **DIAGNOSTICS*, ADAPTIVE MESH=OFF

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Defining ALE Adaptive Mesh Domains in Abaqus/Standard

Products: Abaqus/Standard Abaqus/CAE

References:

- *About ALE Adaptive Meshing*
- *ALE Adaptive Meshing and Remapping in Abaqus/Standard*
- **ADAPTIVE MESH*
- **ADAPTIVE MESH CONSTRAINT*
- **ADAPTIVE MESH CONTROLS*
- *Customizing ALE adaptive meshing*

Overview

ALE adaptive meshing in Abaqus/Standard:

- maintains a topologically similar mesh;
- can be used to solve Lagrangian problems (in which no material leaves the mesh) and to model effects of ablation, or wear (in which material is eroded at the boundary);
- can be used in static stress/displacement analysis, steady-state transport analysis, coupled pore fluid flow and stress analysis, and coupled temperature-displacement analysis;
- can be used only in geometrically nonlinear general analysis steps; and
- is available only for acoustic elements and a subset of the solid elements.

Defining an ALE Adaptive Mesh Domain

You can apply ALE adaptive mesh smoothing to an entire model or to individual parts of the model as a step-dependent feature. Adaptive meshing for solid elements in Abaqus/Standard uses a subset of the adaptive meshing functionality available in Abaqus/Explicit.

You must specify the portion of the original mesh that will be subject to adaptive meshing.

Input File Usage: **ADAPTIVE MESH*, ELSET=*name*

Multiple adaptive mesh domains can be defined in a step by reusing the **ADAPTIVE MESH* option, but each element set must refer to a unique set of elements.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Only one adaptive mesh domain can be defined in Abaqus/CAE for any particular step.

Modifying an ALE Adaptive Mesh Domain

By default, all adaptive mesh domains defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh domains in effect for a given step relative to the preexisting adaptive mesh

domains. At each new step the existing adaptive mesh domains can be modified and additional adaptive mesh domains can be specified (except in Abaqus/CAE, where only one adaptive mesh domain can be in effect for a given step).

Input File Usage: Use either of the following options to modify an existing adaptive mesh domain or to specify an additional adaptive mesh domain:

**ADAPTIVE MESH*, ELSET=*name*
**ADAPTIVE MESH*, ELSET=*name*, OP=MOD

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**

Removing an ALE Adaptive Mesh Domain

If you choose to remove any adaptive mesh domain in a step, no adaptive mesh domains will be propagated from the previous step. Therefore, all adaptive mesh domains that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh domains and to specify new adaptive mesh domains:

**ADAPTIVE MESH*, ELSET=*name*, OP=NEW

If the OP=NEW parameter is used on any **ADAPTIVE MESH* option within a step, it must be used on all **ADAPTIVE MESH* options in the step.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **No ALE adaptive mesh domain for this step**

Splitting ALE Adaptive Mesh Domains

Abaqus/Standard may subdivide each adaptive mesh domain that you specify such that

- all elements in an adaptive domain refer to one element property definition; and
- all elements in an adaptive domain are of similar type (such as hybrid elements with linear pressure).

If Abaqus/Standard subdivides the adaptive mesh domains that you specified, each of the adaptive mesh domain subdivisions will have a new name, which will be used for output and diagnostic purposes. The new names will be formed by concatenating the name of the user-specified element set, a number identifying the subdivision, and the step number. Each of the subdivisions will be further examined to ensure that all the elements in a subdivision are subjected to the same body forces. You may be asked to modify the definition of the adaptive mesh domain to satisfy this requirement.

ALE Adaptive Mesh Regions

Each adaptive mesh domain has an interior region and a boundary region. The boundary region may include distinct kinks that take the form of geometric edges or corners. The nodes on the boundary region are, therefore, further separated into free surface nodes, edge nodes, and constrained nodes. Different updating rules are applied to nodes in these different regions. These regions are created automatically by Abaqus/Standard. You can control the detection of the geometric features. In addition, mesh constraints can be applied to any node in the adaptive mesh domain.

Since acoustic elements do not have displacement degrees of freedom, their treatment for adaptive meshing includes some additional considerations. The acoustic adaptive domain must be connected to the structural

domain using a surface-based tie constraint with the secondary surface defined on the acoustic domain. Thus, an acoustic adaptive domain has an additional boundary region that is connected to the structural domain. These secondary surface nodes are updated based on the displaced configuration of the main surface nodes on the structural domain, without permitting relative sliding between the surfaces. The displacements of the main surface defined on the structural domain, together with nonzero adaptive mesh constraints, serve as the forcing function that drives adaptive mesh smoothing of an acoustic adaptive domain. The mesh smoothing algorithm will produce no changes in the acoustic adaptive domain if these displacements are zero.

Options for controlling the mesh smoothing algorithm are described in *ALE Adaptive Meshing and Remapping in Abaqus/Standard*.

ALE Adaptive Mesh Interior Regions

Nodes in the interior region are defined as nodes that are surrounded entirely by elements in the adaptive mesh domain. By default, the new position of an interior node is computed from the positions of the adjacent nodes that are connected through element edges to the node in question. These nodes can move in any direction.

To control the displacement of these nodes, you can apply an adaptive mesh constraint in any direction.

ALE Adaptive Mesh Boundary Regions

The boundary region is that part of the surface of the adaptive mesh domain that is not constrained to other elements in the mesh. The nodes on the boundary region are further separated into surface nodes, edge nodes, corner nodes, and constrained nodes.

Surface, Edge, and Corner Nodes

Surface nodes are defined as nodes at which the surrounding surface facets have the same normal vector within a user-defined angle. These nodes are constrained against movement in the normal direction, but sliding in any tangential direction is permitted. The new position of a surface node is computed from the positions of the adjacent nodes that are connected through the edges of the surface facets to the node in question.

Edge nodes are nodes in a three-dimensional model at which the surrounding surface facets have two different normals and where the vectors along two of the surface edges are colinear. Nodes on an edge can slide only along the edge. The new position of an edge node is computed from the positions of the two adjacent nodes along the edge.

Corner nodes are nodes at which all the surrounding surface facet normals are different. These nodes are constrained against all mesh smoothing movement.

You can control the displacement of these node types on the boundary region by applying an adaptive mesh constraint in any direction.

Constrained Nodes in an Acoustic Adaptive Domain

A surface-based tie constraint can be used to connect two acoustic surfaces together. When both the main and secondary nodes of the tie constraint belong to the same adaptive mesh domain, the main surface nodes are updated according to the rules for surface, edge, and corner nodes. An adaptive mesh constraint can be applied at main surface nodes. Secondary nodes are updated by applying a tie constraint. Adaptive mesh constraints cannot be applied at secondary surface nodes.

Mesh smoothing is not applied to these nodes when the main and secondary nodes belong to different acoustic adaptive mesh domains.

Constrained Nodes in a Solid Adaptive Domain

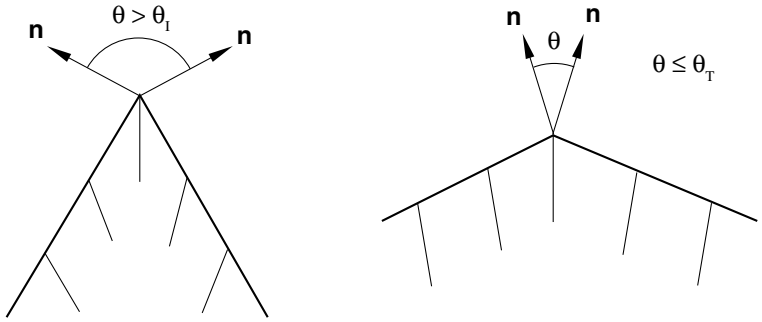
Mesh smoothing is not applied to nodes that are involved in multi-point constraints (see *General Multi-Point Constraints*), equations (see *Linear Constraint Equations*), or kinematic coupling constraints (*Coupling Constraints*).

Geometric Features

The classification of boundary region nodes as surface, edge, and corner nodes is performed based on the identification of geometric features in the mesh's configuration at the start of a step where adaptive mesh domains are defined and is updated as the analysis proceeds and the configuration changes. You can define the criteria that Abaqus/Standard uses in classifying geometric features through adaptive mesh controls.

Controlling the Detection of Geometric Edges and Corners

Geometric features are identified initially as edges on boundary regions where the angle between the normals on adjacent element faces is greater than the initial geometric feature angle, θ_I ($0° \le \theta_I \le 180°$), as shown in *Figure 1*. The default value for the initial geometric feature angle is $\theta_I = 30°$. Setting $\theta_I = 180°$ will ensure that no geometric edges or corners are formed on the boundary of the adaptive mesh domain. You can define adaptive mesh controls to change the value of the angle that will be used to recognize geometric features.



Initial mesh with a geometric feature: no mesh flow is permitted past the corner.

The geometric feature is deactivated during simulation.

Figure 1: Detection and deactivation of geometric features.

Input File Usage: **ADAPTIVE MESH CONTROLS*, NAME=*name*,
INITIAL FEATURE ANGLE=θ_I

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name:** *name*, **Initial feature angle:** θ_I

Controlling the Activation and Deactivation of Geometric Edges and Corners

Abaqus/Standard allows geometric features, and consequently the updating rules applied at a node, to change during the analysis. For example, nodes are constrained to lie along a discrete geometric edge unless the angle forming the geometric edge becomes less than the transition geometric feature angle, θ_T ($0° \le \theta_T \le 180°$). The default value for the transition feature angle is $\theta_T = 30°$. If the angle across the geometric edge becomes less than θ_T, the boundary surface is considered to be flattened sufficiently for the feature to be deactivated, and the mesh is allowed to slide freely on the surface. Geometric corners are allowed to flatten in a similar fashion. In

addition, surfaces that are initially flat might develop edges or corners during the simulation. This logic allows great flexibility in mesh adaptation while preserving geometric features in the model.

Setting $\theta_T = 0°$ will ensure that no geometric edges or corners are ever deactivated. You can change the transition feature angle using adaptive mesh controls.

Abaqus/Standard will issue a warning message when geometric features are activated or deactivated.

Input File Usage: **ADAPTIVE MESH CONTROLS*, NAME=*name*,
TRANSITION FEATURE ANGLE=θ_T

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name:** *name*,
Transition feature angle: θ_T

Mesh Constraints

In most adaptive mesh problems the motion of nodes in the mesh is determined by the mesh smoothing algorithm, with constraints imposed by the domain boundary and the boundary region edges. However, there might be cases when you will want to define the motion of the nodes explicitly. You might also want to keep certain nodes fixed, to move nodes in a particular direction, or to force certain nodes to move with the material.

Adaptive mesh constraints give you the flexibility to define the motion of the node explicitly.

Input File Usage: **ADAPTIVE MESH CONSTRAINT*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Create**

Applying Spatial Mesh Constraints

Spatial mesh constraints are applied to define the motion of the nodes explicitly. Spatial mesh constraints allow full control over the mesh movement and can be applied to any node except those that have Lagrangian mesh constraints applied to them.

You can also prescribe the spatial mesh constraints via user subroutine *UMESHMOTION*. The user subroutine allows you to let the spatial mesh constraints depend on available nodal or material point information.

Input File Usage: Use the following option to define the mesh constraints explicitly:

**ADAPTIVE MESH CONSTRAINT*, CONSTRAINT TYPE=SPATIAL,
TYPE=DISPLACEMENT or VELOCITY

Use the following option to define the mesh constraints in user subroutine *UMESHMOTION*:

**ADAPTIVE MESH CONSTRAINT*, CONSTRAINT TYPE=SPATIAL,
TYPE=DISPLACEMENT or VELOCITY, USER

Abaqus/CAE Usage: To define the mesh constraints explicitly:

Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**

To define the mesh motion in user subroutine *UMESHMOTION*:

Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: User-defined**

Defining Mesh Constraints That Vary with Time

The prescribed magnitude of a nonzero mesh constraint can vary with time during a step according to an amplitude definition (see *Amplitude Curves*).

Input File Usage: Use both of the following options:

**AMPLITUDE*, NAME=*name*
**ADAPTIVE MESH CONSTRAINT*, AMPLITUDE=*name*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**: **Amplitude:** *amplitude*

Applying Spatial Mesh Constraints in Local Directions

Mesh constraints are applied in local directions if a transformed coordinate system is used at a node (*Transformed Coordinate Systems*); otherwise, they are applied in global directions.

Applying Lagrangian Mesh Constraints

Lagrangian mesh constraints on a node are used to indicate that mesh smoothing should not be applied; that is, the node must follow the material.

Input File Usage: **ADAPTIVE MESH CONSTRAINT*,
CONSTRAINT TYPE=LAGRANGIAN

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Follow underlying material**

Spatial Mesh Constraint Considerations

When you decide on the type of spatial adaptive mesh constraint, (displacement, velocity, or specified with a user subroutine), you should consider the guidelines below.

Choosing between Displacement and Velocity Adaptive Mesh Constraints

Displacement and velocity mesh constraints differ in their application. Displacement constraints define a node's displacement relative to its original coordinates, while velocity constraints define a node's velocity relative to the motion of the material. You will use a displacement constraint to control a node's motion to a specific coordinate location, while you will use a velocity constraint to control a node's motion relative to the Lagrangian motion. Therefore, a constant velocity adaptive mesh constraint does not in general lead to a constant velocity of the node relative to its original coordinates.

Applying Spatial Adaptive Mesh Constraints to Model Material Ablation

Your spatial mesh constraint is applied without regard to the current material displacement at the node. This behavior allows you to prescribe mesh motion that differs from the current material displacement at the free surface of the adaptive mesh domain, effectively eroding, or adding, material at the boundary. Using adaptive mesh constraints this way is an effective technique for modeling wear or ablation processes. As described above, in common ablation modeling cases you will use the velocity form of the constraint. In addition, for general boundary shapes the most effective interface for ablation is user subroutine *UMESHMOTION*, where you can apply spatial mesh constraints to the nodes on the free surface in general ways according to solution-dependent variables, if needed. The user subroutine interface provides a local coordinate system that is normal to the free surface at the surface node, enabling you to describe mesh motions in this local system.

Modifying ALE Adaptive Mesh Constraints

By default, all adaptive mesh constraints defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh constraints in effect for a given step relative to the preexisting adaptive mesh constraints. At each new step, the existing adaptive mesh constraints can be modified and additional adaptive mesh constraints can be specified.

Input File Usage: Use either of the following options to modify an existing adaptive mesh constraint or to specify an additional adaptive mesh constraint:

**ADAPTIVE MESH CONSTRAINT*,
**ADAPTIVE MESH CONSTRAINT*, OP=MOD

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Manager**: select the desired step and mesh constraint: **Edit**

Removing ALE Adaptive Mesh Constraints

If you choose to remove any adaptive mesh constraint in a step, no adaptive mesh constraints will be propagated from the previous step. Therefore, all adaptive mesh constraints that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh constraints and to specify new adaptive mesh constraints:

**ADAPTIVE MESH CONSTRAINT*, OP=NEW

If the OP=NEW parameter is used on any **ADAPTIVE MESH CONSTRAINT* option within a step, it must be used on all **ADAPTIVE MESH CONSTRAINT* options in the step.

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Constraint->Manager**: select the desired step and mesh constraint: **Deactivate**

Contact

When surfaces are defined for large-sliding contact, adaptive meshing might relocate the nodes on the surfaces. If the bodies in contact are sliding or deforming considerably, you might want to use Lagrangian mesh constraints on the boundary of the surfaces to prevent the surfaces from sliding from their intended place.

For small-sliding contact Abaqus/Standard assumes that the reference configuration does not change significantly. If the reference configuration does not change significantly, the amount of adaptive meshing on these surfaces

should be small and the contact quantities computed based on the reference configuration should continue to remain valid (Abaqus/Standard updates the tangent planes if nodes change positions). Hence, Abaqus/Standard will allow the nodes on the contact surface to move as needed by the mesh smoothing. You should apply Lagrangian mesh constraints in cases where nodes are intended to remain nonadaptive.

Initial Conditions

Initial temperatures and field variables can be defined on any region subjected to adaptive mesh smoothing. However, these variables will not be remapped from the original to the updated configuration.

Loads

For elements with displacement degrees of freedom, no restrictions are made to loads applied to adaptive mesh domains. In cases where loads are intended to follow the material motion, Lagrangian mesh constraints must be applied to the nodes on the boundary of the surface on which distributed loads are applied to prevent the surface from sliding. This will allow adaptive meshing to occur inside the surface while maintaining the location of the distributed load.

All the nodes on which concentrated loads are applied become nonadaptive.

The loads that can be applied to an acoustic domain are described in *Coupled Acoustic-Structural Analysis*. These loads cannot be applied in procedures in which mesh smoothing can be performed.

Boundary Conditions

Special consideration is given to nodes on which boundary conditions are applied. No adaptive meshing is done in the direction in which the boundary condition is applied, but adaptive meshing is carried out in other directions. When a boundary condition on any degree of freedom is removed (see *Boundary Conditions*) in a step, the node becomes nonadaptive, since Abaqus/Standard will ramp off the corresponding reaction force or flux over the duration of the step. Boundary conditions on scalar degrees of freedom will cause the node to become nonadaptive.

The boundary conditions that can be applied to an acoustic domain are described in *Coupled Acoustic-Structural Analysis*. These boundary conditions cannot be applied in any analysis procedure in which mesh smoothing can be performed.

Predefined Fields

There are no restrictions on applying prescribed temperatures or field variables in an adaptive mesh domain, but these nodal values are not remapped when adaptive meshing is performed. Therefore, predefined fields that are not constant might not be meaningful in an adaptive mesh domain.

Material Options

For elements with displacement degrees of freedom all material models that are isotropic and homogeneous can be used in an adaptive domain. Material options that have anisotropic behavior such as anisotropic materials (see *Defining Fully Anisotropic Elasticity*), jointed material models (see *Jointed Material Model*), and concrete material models (see *Concrete Smeared Cracking*) cannot be used in an adaptive mesh domain.

For acoustic elements the relevant material models are described in *Coupled Acoustic-Structural Analysis*. Mesh smoothing assumes that the geometric changes in the acoustic domain do not lead to changes in material properties, such as fluid density.

Elements

Adaptive mesh domains can be defined for all acoustic first-order and second-order planar, axisymmetric, and three-dimensional elements in Abaqus/Standard and for a limited number of other elements. *Table 1* provides a list of supported elements.

Table 1: Elements supported for adaptive meshing.

AC1D2, AC1D3, AC2D3, AC2D4, AC2D6, AC2D8, AC3D4, AC3D6, AC3D8, AC3D10, AC3D15, AC3D20, ACAX3, ACAX4, ACAX6, ACAX8
CPS4, CPS4T, CPS3
CPE4, CPE4H, CPE4T, CPE4HT, CPE4P, CPE4PH, CPE3, CPE3H
CAX4, CAX4H, CAX4T, CAX4HT, CAX4P, CAX4PH, CAX3, CAX3H
C3D8, C3D8R, C3D8H, C3D8RH, C3D8T, C3D8HT, C3D8RT, C3D8RHT, C3D8P, C3D8PH, C3D8RP, C3D8RPH

Procedures

Adaptive meshing can be used only in geometrically nonlinear general steps that invoke one of the following procedures:

- Static stress/displacement analysis (*About Static Stress Analysis Procedures*)
- Steady-state transport analysis (*Steady-State Transport Analysis*)
- Coupled temperature-displacement analysis (*Fully Coupled Thermal-Stress Analysis*)
- Coupled pore fluid flow and stress analysis (*Coupled Pore Fluid Diffusion and Stress Analysis*)

Acoustic elements will typically undergo adaptive meshing during static procedures and then participate in subsequent acoustic procedures in their updated configuration.

Limitations

- Elements within the adaptive domain cannot be removed or added (*Element and Contact Pair Removal and Reactivation*).
- Deformable elements that are declared rigid cannot be part of adaptive mesh domains.
- Elements in the adaptive domain cannot contain embedded elements or rebars.
- Symmetric results transfer cannot be done from an axisymmetric model that had solid elements in an adaptive domain.
- Import cannot be done from a model that had solid elements in the adaptive domain.
- It is not meaningful to drive a submodel using the nodes from a global model that were part of an adaptive mesh domain.
- Only enhanced hourglass control can be used with reduced-integration elements.
- When used with acoustic elements, adaptive mesh smoothing must be applied in steps prior to a coupled structural-acoustic analysis. It cannot be applied during a large-displacement dynamic analysis.
- Mesh smoothing assumes that the geometric changes in the acoustic domain do not lead to changes in material properties, such as fluid density.
- The coupling between the fluid and structure must be defined using a surface-based tie constraint with the secondary surface defined on the acoustic domain.
- Nodes in the adaptive domain that are involved in constraints such as multi-point constraints (*General Multi-Point Constraints*) and equations (*Linear Constraint Equations*) should be made nonadaptive by applying Lagrangian constraints.

Input File Template

Applying ALE Adaptive Meshing for Acoustic Analysis

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define structural elements
*SURFACE, NAME=TIE_ACOUSTIC
Data lines to define the acoustic surface interface with the structural mesh
*SURFACE, NAME=TIE_SOLID
Data lines to define the solid surface interface with the acoustic mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
…
*STEP
*STATIC
*ADAPTIVE MESH, ELSET=ACOUSTIC, MESH SWEEPS=10
…
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, DIRECT
…
*END STEP
```

Applying ALE Adaptive Meshing in Other Uses

```
*HEADING
…
*ELEMENT, TYPE=C3D8, ELSET=..
Data lines to define solid elements
*NSET, NSET=LAG
Data lines to define nodes that should be nonadaptive
*NSET, NSET=SPATIAL
Data lines to define nodes that will have spatial adaptive mesh constraints applied
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define structural elements
*STEP, NLGEOM=YES
*STATIC
*ADAPTIVE MESH, ELSET=SOLID, MESH SWEEPS=10
*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=LAGRANGIAN
LAG
*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=SPATIAL, USER
SPATIAL
*END STEP
```

ALE Adaptive Meshing and Remapping in Abaqus/Standard

Products: Abaqus/Standard Abaqus/CAE

References:

- *Defining ALE Adaptive Mesh Domains in Abaqus/Standard*
- **ADAPTIVE MESH*
- **ADAPTIVE MESH CONSTRAINT*
- **ADAPTIVE MESH CONTROLS*
- *Customizing ALE adaptive meshing*

Overview

ALE adaptive meshing consists of two fundamental tasks:

- creating a new mesh through a process called sweeping, and
- remapping solution variables from the old mesh to the new mesh with a process called advection.

You can control the process of mesh sweeping after which, if necessary, Abaqus/Standard will automatically perform advection. The default methods for creating a new mesh have been chosen carefully to work for acoustic analysis and for modeling the effects of ablation, or wear, of material. However, you might need to override the default choices to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing for other types of applications.

Adaptive mesh smoothing is defined as part of a step definition. The adaptive meshing in Abaqus/Standard uses an operator split method wherein each analysis increment consists of a Lagrangian phase followed by an Eulerian phase. The Lagrangian phase is the typical Abaqus/Standard solution increment where neither mesh sweeps nor advection occur. Once the equilibrium equations have converged, mesh smoothing is applied. Following the adjustment of nodes through the mesh sweeping process, material point quantities are advected in an Eulerian phase to account for the revised meshing of the model in its current configuration. This operator split method is chosen to avoid unsymmetric Jacobian terms that would result when the advection and material straining occur simultaneously. Advection is not required for, and is not applied to, acoustic elements.

The ALE Adaptive Mesh Sweeping Algorithm

Adaptive mesh smoothing is performed after the structural equilibrium equations have converged. The mesh smoothing equations are solved explicitly by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated—based on the positions of neighboring nodes obtained during the previous mesh sweep—to reduce element distortion. The new position, $\mathbf{x}_{i+1}$, of a node is obtained as

$$\mathbf{x}_{i+1} = \mathbf{X} + \mathbf{u}_{i+1} = \mathbf{N}^N \mathbf{x}_i^N,$$

where $\mathbf{X}$ is the original position of the node, $\mathbf{u}_{i+1}$ is the nodal displacement, $\mathbf{x}_i^N$ are the neighboring nodal positions obtained during the previous mesh sweep, and $\mathbf{N}^N$ are weight functions obtained from one or a weighted mixture of the following methods. The displacements applied during sweeps are not associated with mechanical behavior.

Original Configuration Projection

Original configuration projection is the default in Abaqus/Standard and determines the weight function from a least squares minimization procedure that minimizes node displacement in a projection of the mesh back to the original configuration. This method of smoothing affects only deformations of the mesh and not the original mesh.

Volume Smoothing

Volume smoothing determines the weight function by computing a volume-weighted average of the element centers in the elements surrounding the node. In *Figure 1* the new position of node M is determined by a volume-weighted average of the positions of the element centers, C, of the four surrounding elements. The volume weighting will tend to push the node away from element center C1 and toward element center C3, thus reducing element distortion.

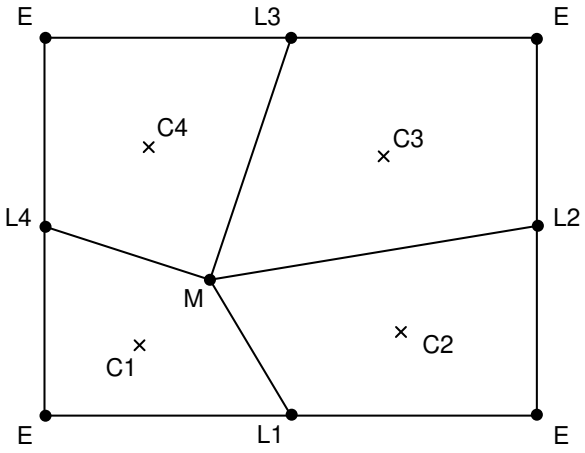


Figure 1: Relocation of a node during a mesh sweep.

Volume smoothing is supported in structured domains, where every node is surrounded by four elements in two dimensions or eight elements in three dimensions.

Combining Smoothing Methods

The default smoothing method in Abaqus/Standard is original configuration projection. To choose an alternate smoothing method or to combine the smoothing methods, you specify the weighting factor for each method. When more than one smoothing method is used, a node is relocated by computing a weighted average of the locations predicted by each chosen method. All weights must be zero or positive, and their sum must be nonzero. The weights are significant only in a relative sense; their values are normalized so that their sum is 1.0.

Input File Usage: **ADAPTIVE MESH CONTROLS*, NAME=*name*
original configuration projection weight, volume smoothing weight

For example, the following option could be used to define an equal blend of original configuration projection and volume smoothing:

**ADAPTIVE MESH CONTROLS*, NAME=*name*
0.5, 0.5

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, **Original configuration projection:** *original configuration projection weight*, **Volumetric:** *volume smoothing weight*

Geometric Enhancements to the Basic Smoothing Methods

The conventional forms of the basic smoothing methods might not perform well in highly distorted domains. You can use geometrically enhanced forms of the basic smoothing algorithms as a technique to mitigate distortion. These forms are heuristic and based on nodal locations only. Due to their heuristic nature, geometric enhancements might not always improve the mesh smoothing.

Input File Usage: Use the following option to apply geometric enhancements to the smoothing algorithm:

**ADAPTIVE MESH CONTROLS*, NAME=*name*,
GEOMETRIC ENHANCEMENT=YES

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Controls->Create**: **Name**: *name*, toggle on **Use enhanced algorithm based on evolving element geometry**

Application of the Sweeping Algorithm

The mesh smoothing process begins with the mesh in its current displaced equilibrium configuration. Nodes that have no displacement degrees of freedom, such as those connected to acoustic elements, are maintained at their most recent configuration. Mesh smoothing is then driven by distortions in the current configuration and by boundary constraints. These boundary constraints can be described directly through adaptive mesh constraints. In the case of structural-acoustic boundaries the structural mesh boundary provides a constraint that controls the smoothing of adjacent acoustic element regions.

When these boundary constraints are much larger than the characteristic element length in the adaptive mesh domain, significant geometric changes, such as the development of corners, can occur. To prevent such changes, the constraints are applied gradually over a series of "subincrements" onto the domain boundary. The number of subincrements used is determined on the basis of the magnitude of the maximum surface displacement and the characteristic element dimension.

The remaining nodes (nodes not driven by constraints) are identified as interior nodes, free surface nodes, edge nodes, or corner nodes. These nodes are treated as described in *Defining ALE Adaptive Mesh Domains in Abaqus/Standard*.

At the end of mesh sweeping the new geometry is checked to ensure that elements did not become severely distorted during mesh smoothing. Abaqus/Standard responds to severe distortion in different ways, depending on the elements and procedures used. When adaptive meshing is used with acoustic elements, the current analysis increment is repeated with a reduced time increment, followed by another adaptive mesh smoothing attempt. When adaptive meshing is used with other elements, severe distortion results in abandonment of mesh smoothing for that increment. In cases where adaptive mesh constraints are also defined, Abaqus/Standard exits since the constraints cannot be satisfied.

Controlling the Frequency of ALE Adaptive Mesh Smoothing

In most cases the frequency of adaptive meshing is the parameter that most affects the mesh quality. By default, mesh smoothing will be performed after each converged structural analysis increment. You can change the frequency of adaptive meshing, except when spatial adaptive mesh constraints are defined.

Input File Usage: **ADAPTIVE MESH*, FREQUENCY=*number of increments*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the adaptive mesh domain below**, **Frequency:** *number of increments*

Controlling Convergence of ALE Adaptive Mesh Smoothing

The adaptive mesh smoothing equations are solved explicitly by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated based on the current positions of neighboring nodes to reduce element distortion.

Mesh smoothing is performed following the end of a converged increment. You can control the intensity of the mesh smoothing by defining the number of mesh sweeps required. When the displacements are large, more iterations are usually required. When used in acoustic analyses, more iterations are usually required when the volume of the elements in the acoustic domain decreases compared to the case when the volume increases during structural loading.

You can specify the number of mesh sweeps to be performed in each adaptive mesh increment. The default number of mesh sweeps is five.

By applying the mesh sweeping algorithm repeatedly, the mesh will converge; in other words, nodal positions are obtained that do not change with further mesh sweeping. However, it is usually not necessary to apply mesh smoothing until a converged mesh is obtained; the main objective is to reduce element distortion.

Input File Usage: **ADAPTIVE MESH*, MESH SWEEPS=*number of sweeps*

Abaqus/CAE Usage: Step module: **Other->ALE Adaptive Mesh Domain->Edit**: toggle on **Use the adaptive mesh domain below**, **Remeshing sweeps per increment:** *number of sweeps*

Controlling Mesh Distortion of ALE Adaptive Mesh Smoothing

The default settings for the adaptive mesh smoothing algorithms described in earlier sections work well in most cases. However, there might be situations where the default settings lead to a distorted mesh. In these situations, a better quality mesh might be obtained by one or more of the techniques described earlier. These include the following:

- increasing the number of mesh sweeps, and/or
- combining smoothing methods by specifying individual weights on the original configuration projection and the volume smoothing methods, and/or
- activating geometric enhancements to the basic smoothing methods.

The ALE Adaptive Mesh Advection Algorithm

Abaqus/Standard applies an explicit method, based on the Lax-Wendroff method, to integrate the advection equation. The key principle of the Lax-Wendroff method is replacement of the time derivatives of the material point quantities with the spatial derivatives using the classical relationship between the material time derivative, the referential derivative, and the spatial derivative. The update scheme is second-order accurate and provides some upwinding. Nodal quantities are advected by first converting them to the material point quantities.

Advection of the material quantities will generally result in loss of equilibrium, for two main reasons. The first reason is the errors in the advection process itself. To minimize the errors in advection, Abaqus/Standard imposes restrictions on the magnitude of the advection velocity by requiring that the Courant number for every element in the adaptive domain be less than one. In cases where the Courant number is greater than one you will be informed and Abaqus/Standard will generate multiple advection passes per increment. The second reason for the loss of equilibrium is changes in the representation of the underlying material quantities by the changed

mesh. For example, consider a region of the structure having some stress gradients spanned initially by two elements. After mesh smoothing, the same region might have more than two elements. This will lead to slightly different volume integration while computing the internal force even when there are no errors in advection.

These sources of error in equilibrium are significant only when the mesh is too coarse to provide a good solution and mesh smoothing is carried out with such small frequency that the mesh motion is larger than the average element size. In practical applications these errors are typically insignificant, the resulting loss of equilibrium is generally small, and the residuals generated by the loss of the equilibrium fall within the limits of the Abaqus/Standard convergence criterion. Any loss of equilibrium is not propagated since equilibrium will again be satisfied at the end of the Lagrangian phase of the next increment.

Impact of Advection on Subsequent Steps

To ensure that the results are output only for the configuration that satisfies equilibrium, Abaqus/Standard always outputs the results at the end of the Lagrangian phase. The Eulerian phase that follows the Lagrangian phase will leave the structure out of equilibrium for the next increment. This sequence has a consequence that after the last Eulerian phase is carried out at the end of the step, equilibrium will not be satisfied exactly at the beginning of the next step and the solution at the end of the step will differ slightly from the solution at the zero increment of the following step. Equilibrium can again be established by following the step that had adaptive meshing by a step that removes all the adaptive mesh domains and allows the structure to equilibrate. A one-increment step will usually suffice. This is particularly important when the following step is a perturbation procedure that uses the solution from the previous step as the base state.

Frequency steps that follow adaptive mesh steps will also be impacted, because element mass is not advected during mesh smoothing. This impact on the element mass can be significant, depending on the extent of adaptive mesh motion and change in element size due to mesh smoothing. Abaqus will provide a warning message in cases where adaptive meshing precedes a frequency step; you should evaluate the impact of your updated mesh configuration when interpreting results from a frequency step in these cases.

Output

In adaptive meshing the integration point of an element will generally not refer to the same material point throughout the analysis. Contour plots of material variables will show correct spatial distribution, but history plots are not meaningful. The displacement of the nodes contains the material displacement as well as the displacement due to mesh motion. You can obtain measures of the volume lost due to adaptive mesh constraints with the partial model variable VOLC, which is useful when using adaptive mesh constraints to model ablation.

A summary of the adaptive meshing in each adaptive mesh domain is written to the message (`.msg`) file. This summary includes the total number of load increments over which the structural displacement is transferred to the fluid, the total number of mesh sweeps performed, the magnitude of the maximum displacement increment, and the node and degree of freedom at which the maximum displacement increment is measured. Warning messages are issued when geometric features change during mesh smoothing.

More detailed diagnostic output for adaptive mesh smoothing can be requested; see *The Abaqus/Standard Message File*. This output provides the magnitude of the maximum displacement and the node and degree of freedom where the maximum displacement increment occurs during each mesh sweep. In addition, the nodes experiencing changes in geometric features are listed.

References

- Lax, P. D., and B. Wendroff, “Difference Schemes for Hyperbolic Equations with High-Order Accuracy,” *Communications on Pure and Applied Mathematics*, vol. 17 381, 1964.
- Lax, P. D., and B. Wendroff, “Systems of Conservations Laws,” *Communications on Pure and Applied Mathematics*, vol. 13, pp. 217–237, 1960.

Adaptive Remeshing

In this section:

- *About Adaptive Remeshing*
- *Selection of Error Indicators Influencing Adaptive Remeshing*
- *Solution-Based Mesh Sizing*

About Adaptive Remeshing

Overview

Abaqus/CAE provides an automated process to remesh your model adaptively.

The goal of the adaptive remeshing process is to approach or reach targets on selected error indicators for a specified model and its accompanying load history. See *About Adaptivity Techniques* for a comparison of this process to other Abaqus adaptivity methods.

Overview

The following steps are required to incorporate adaptive remeshing into your Abaqus/CAE model:

- You identify regions of the model where you wish to apply one or more adaptive remeshing rules. A remeshing rule defines the step during which it will be applied, the error indicator output variables and targets for those error indicators, the sizing method, and any constraints on element size. See *What are remeshing rules?*.
- You define a succession of analysis jobs, an "adaptivity process," that will be run as Abaqus/CAE attempts to meet your remeshing rule targets. See *What is an adaptivity process?*.

Based on these remeshing rules and your adaptivity process definition, Abaqus/CAE iteratively:

- executes an Abaqus/Standard analysis, which will write selected error indicator output variables based on your remeshing rule settings (See *Selection of Error Indicators Influencing Adaptive Remeshing*),
- uses the error indicator variables in a sizing function to compute element sizes for a new mesh, respecting any size constraints you might specify (See *Solution-Based Mesh Sizing*), and
- generates a new mesh in the regions specified, based on the computed element sizes. The neighboring regions will also be remeshed.

These iterations continue until either:

- all remeshing rule targets are satisfied, or
- a maximum number of remesh iterations is reached.

See *When will my mesh adaptivity stop iterating?* for more details. *Figure 1* shows the interaction of Abaqus products and files in this process.

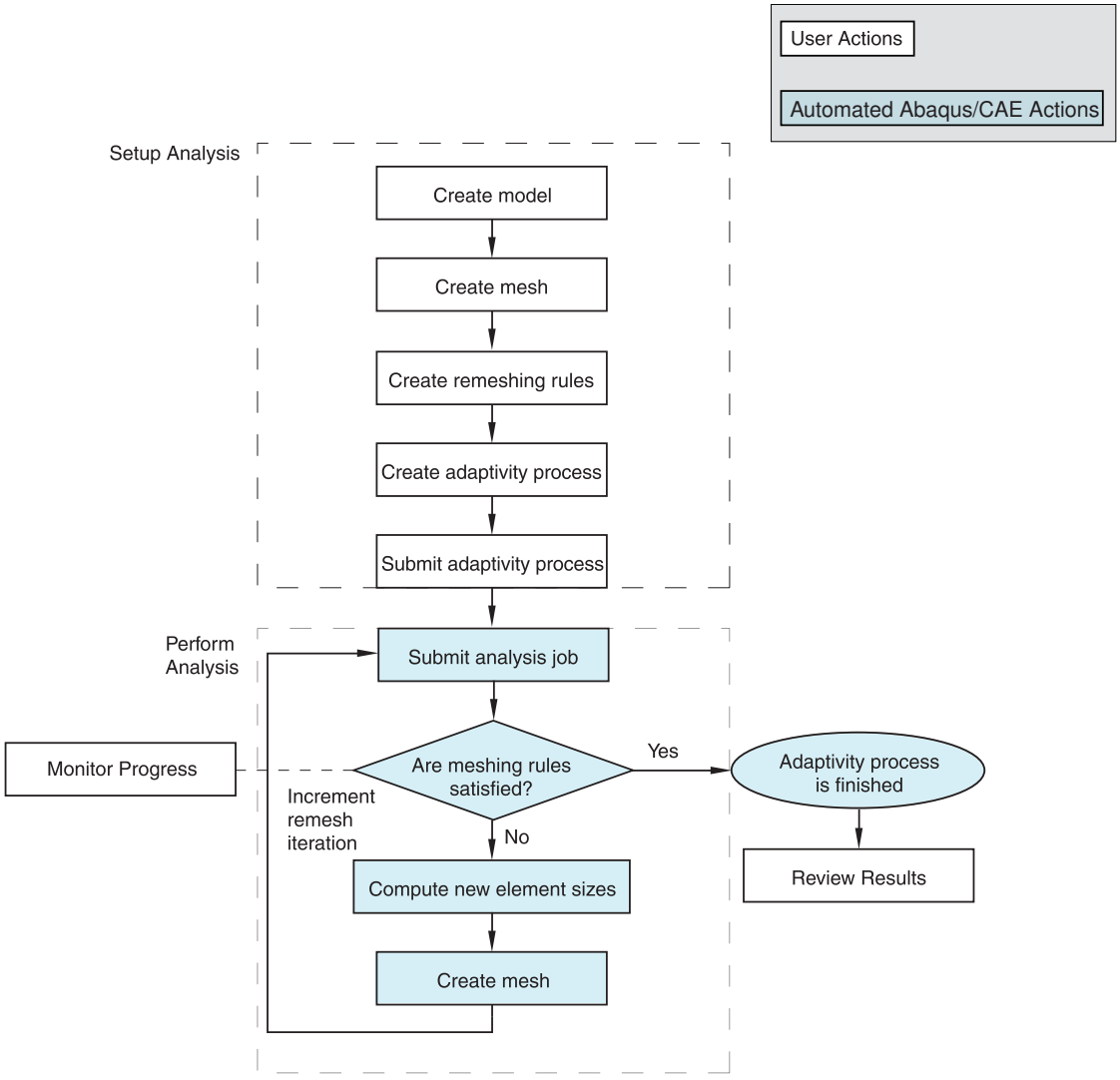


Figure 1: User actions and automated Abaqus/CAE actions in the adaptive remeshing process.

Typical Applications

Adaptive remeshing can improve the quality of your simulation results. Adaptive remeshing can be helpful when:

- you are unsure how refined a mesh needs to be to reach a particular level of accuracy or how coarse the mesh can be without unacceptably impacting solution accuracy;
- it is difficult to design an adequately refined mesh near a region of interest, such as near a stress riser; or
- you do not know a location of interest, such as with formation of a plastic zone, a priori.

An example of using adaptive remeshing to study the thermal and stress behavior of a bolted vessel is provided in *Thermal-stress analysis of a reactor pressure vessel bolted closure*. The example includes a Python script that you can run from Abaqus/CAE to create the model and the remeshing rules. A second script allows you to submit the adaptivity process and to view the changing mesh as Abaqus/CAE computes new element sizes.

Example: Stress Riser

Figure 2 shows how adaptive remeshing generates a high-quality mesh for a typical notched specimen subjected to axial loading.

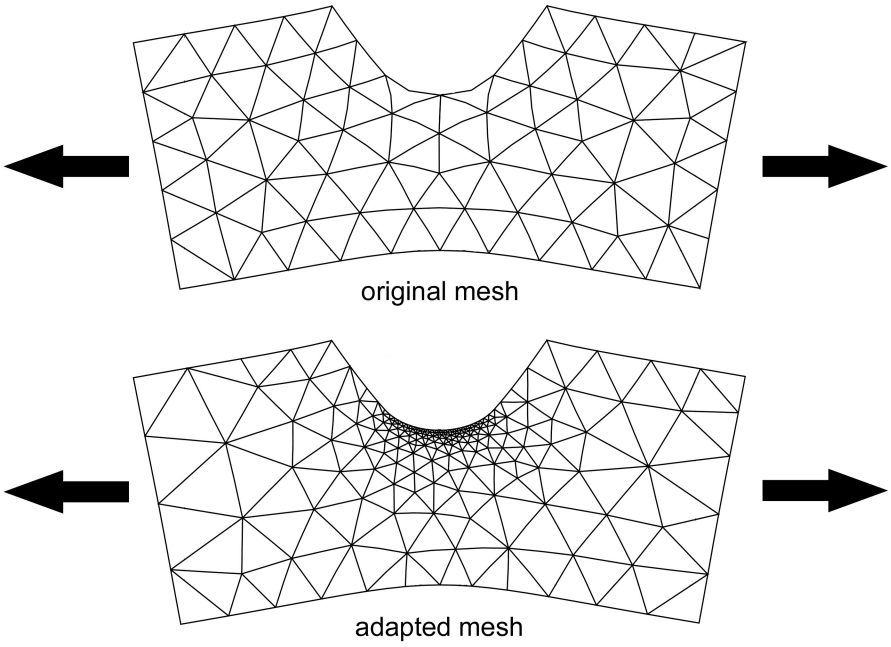


Figure 2: Stress riser mesh before and after refinement.

Figure 3 shows the effect of these mesh changes on solution accuracy in comparison to the effect of uniform mesh refinement on solution accuracy. Adaptive mesh refinement is much more efficient than uniform mesh refinement at reducing solution error.

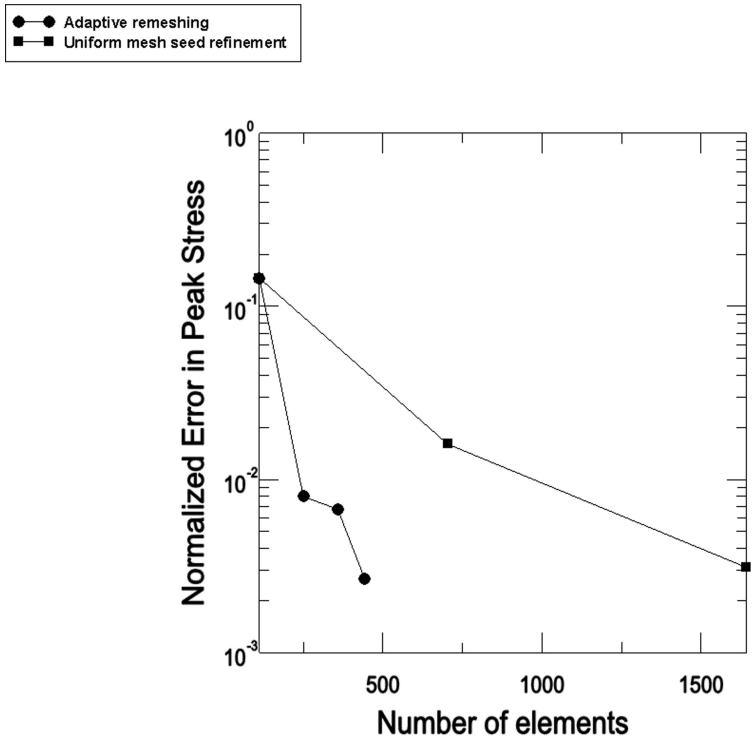


Figure 3: Comparison of adaptive remeshing to uniform mesh refinement based on boundary seeding.

Example: Plastic Hinge

This example, a doubly-notched specimen axially strained until a plastic hinge or band forms, is used to demonstrate how adaptive remeshing will focus a mesh on a plastic hinge. It illustrates the value of adaptive remeshing in cases where the region of interest may not be known a priori. *Figure 4* shows the specimen and the region of active yielding. *Figure 5* shows the original mesh and the adapted mesh after three adaptive remeshing iterations.

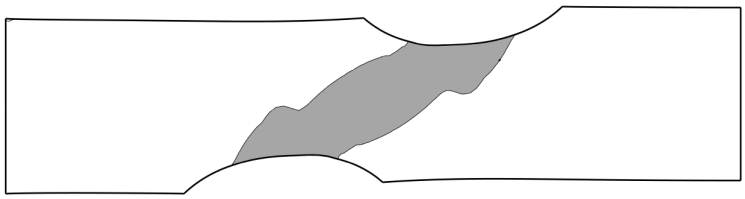


Figure 4: Region of active yielding in a doubly-notched specimen.

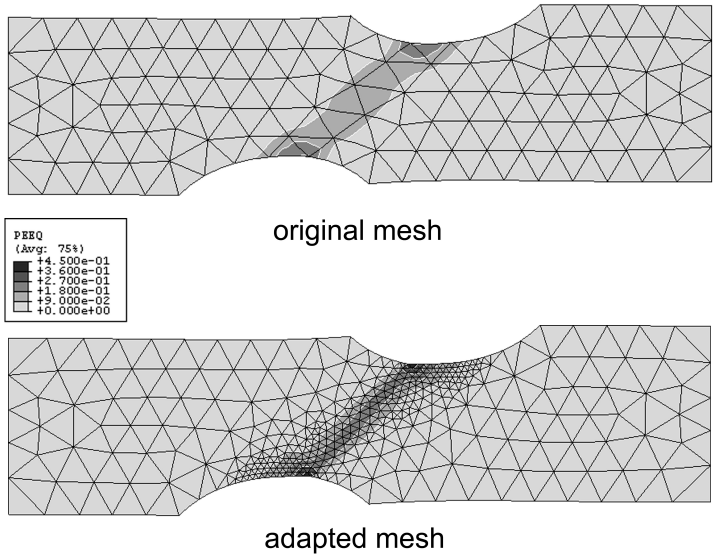


Figure 5: Mesh of doubly-notched specimen before and after adaptive remeshing.

Preparing Your Model for Adaptive Remeshing

You use Abaqus/CAE to do the following when performing adaptive remeshing:

- create the model and specify the boundary conditions and loading history,
- create remeshing rules,
- create an adaptivity process, and
- start and monitor the progress of the adaptivity process.

Creating the Model

You do not have to consider adaptive remeshing when you create the model and specify the boundary conditions and loading history; however, before using adaptive remeshing you must do the following:

- create the geometry of the model—you cannot use an orphan mesh part—and
- provide an initial, nominal, mesh. This mesh can be fairly coarse. Providing an extremely coarse mesh, however, can result in more adaptive remesh iterations due to the poor quality of early remesh iteration error indicator calculations. You can, in typical cases, define a reasonable initial mesh by using the default part instance mesh seeding in Abaqus/CAE.

Creating a Remeshing Rule

You create and configure a remeshing rule using the Mesh module in Abaqus/CAE. See *Creating a remeshing rule* for details on defining remesh rules. Refer to *Selection of Error Indicators Influencing Adaptive Remeshing* and *Solution-Based Mesh Sizing* for details on the methods used to determine revised mesh size distributions.

Abaqus/CAE Usage: Mesh module: **Adaptivity**->**Remeshing Rule**->**Create**

Creating an Adaptivity Process

You create and configure an adaptivity process using the Job module in Abaqus/CAE. When you create an adaptivity process, you can specify the maximum number of remesh iterations to be performed and set various system resource parameters. See *Creating, editing, and manipulating jobs* for details.

Abaqus/CAE Usage: Job module: **Adaptivity->Create**

Performing Adaptive Remeshing with a Provisional Analysis

In some cases you will want to determine an adequate mesh for your model prior to conducting a fully detailed analysis, which might include many steps and complex behavior. A “provisional” analysis can often be used, along with adaptive remeshing, to efficiently determine a good mesh for a model. The provisional analysis may include various simplifications of your fully detailed analyis, such as

- replacing your steps with a single linear perturbation step with loading that adequately reflects your more general loading cases,
- removing plasticity and other material nonlinearities, and
- disabling geometric nonlinearity.

The provisional analysis approach may result in a mesh that is not ideally suited to your ultimate choice of loading. However, the cost for obtaining a mesh from a provisional model may be significantly lower than the case where your adaptivity process considers all of the complexity in the fully detailed analysis, and you may find the refined mesh adequate for use in a variety of analysis situations.

Special Considerations

In general, the Abaqus adaptive remeshing process iterates automatically toward a better quality mesh; however, you should be aware of certain considerations.

Singularities

Stress singularities frequently result from geometric abstractions, such as reentrant corners and contact of a sharp edge in elastic materials, and from point loads or abruptly ended distributed load regions. In these situations the stress field near the singularity is unbounded, and no amount of mesh refinement will enable resolution of the correct solution. If you apply the adaptive remeshing process to regions of your model that include singularities, the process will drive elements near the singularity to very small sizes. The end result may be unacceptably expensive analyses.

You can prevent excessively expensive analyses of models with singularities using the following techniques:

- Exclude the region of the singularity from consideration in the remeshing process. You exclude a region by partitioning the model and assigning remeshing rules only to regions away from the singularity.
- Apply a minimum element size constraint in the remeshing rule. Abaqus/CAE does assign a minimum element size by default, which is a fraction of the default part instance mesh seed. You can modify this constraint to achieve a quality solution near the singularity while avoiding an excessively refined mesh. You can also use the remeshing rule to control the rate at which Abaqus/CAE refines the size of the elements. Element size constraints may prevent an adaptivity process from achieving specified error indicator targets.
- Specify a maximum number of elements for a remeshing rule region. Abaqus/CAE adjusts the mesh sizing such that the generated total number of elements approximately satisfies this constraint.

Convergence Issues

Figure 6 shows a typical history of an error indicator and the computational cost, in Abaqus/Standard, versus remesh iteration.

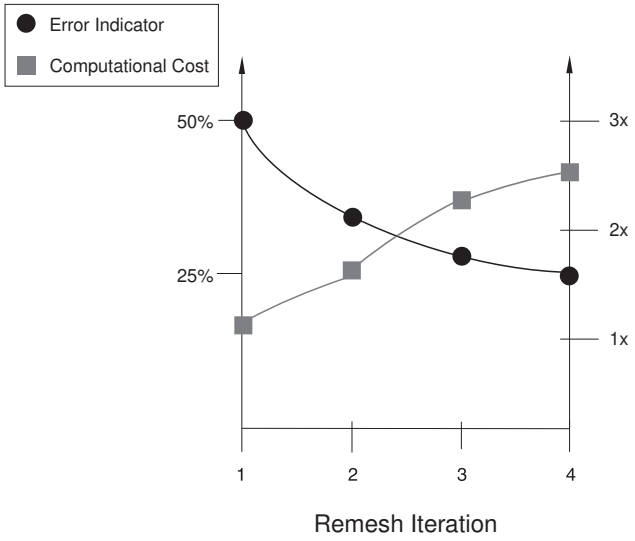


Figure 6: Error indicator and computational cost versus iteration for a model with a 25% error indicator target.

The example in *Figure 6* shows a desirable convergence profile. The solution error indicator decreases monotonically and quickly to the desired 25% error indicator target. Accompanying this error indicator decrease is a moderate increase in computational cost, measured either in model degrees of freedom or time in Abaqus/Standard. Certain situations can interfere with this desirable convergence profile, as follows:

- If your initial mesh is too coarse, the error indicator variables may be of insufficient quality to result in a mesh that is sufficiently improved in the next iteration. The adaptive remeshing process typically creates a high-quality mesh eventually even if the initial mesh is quite coarse. However, some mesh iterations can be avoided with a reasonably refined initial mesh.
- Minimum element size constraints and constraints on the maximum number of elements that you specify when creating the remeshing rule can prevent the mesh from achieving sufficient refinement (in the extreme case of singularities this will always be the case) to satisfy your error indicator targets. You may be able to satisfy your targets by relaxing these constraints; for example, by decreasing the minimum element size. For more information, see *What are remeshing rules?*.
- In addition to producing small mesh sizes resulting in a large number of elements, singularities can cause an adaptivity process to fail in achieving the error target or to require more remeshing iterations. As described in *Singularities*, above, you can control the computational cost by specifying a minimum element size constraint or the maximum number of elements. In any case where a singularity exists within a remeshing rule region, you may see poor convergence in the error indicator results.
- Linear elements (C3D4, CPS4, etc.) and modified elements (C3D10M, CPS6M, etc.) converge slowly compared to quadratic elements (C3D10, CPS6, etc.) requiring a relatively large number of elements to achieve a given error target. Hence, you should use quadratic elements whenever possible.

Continuing a Stopped Adaptive Remeshing Process

The adaptive remeshing process is designed to be automatic—Abaqus/CAE performs a sequence of analyses as it continues to refine your mesh. However, there are occasions where the process will stop and you will want to continue adaptive remeshing from your most recent mesh:

- when you want to change remeshing rules for later remesh iterations, or
- when the adaptive remesh process fails to complete due to machine resource problems.

You can continue the adaptive remeshing process by resubmitting an existing adaptivity process, creating and submitting a new adaptivity process, or performing manual remeshing. See *Manually resizing and remeshing*.

Limitations

Adaptive remeshing requires the use of Abaqus/CAE, and only Abaqus/Standard procedures are supported. Other specific limitations also apply.

Element Types

Abaqus/CAE can perform adaptive remeshing only with elements of the following shapes (see *Which mesh controls can I use with adaptive remeshing?*):

- Planar continuum triangles and quadrilaterals
- Shell triangles and quadrilaterals
- Tetrahedrals

Procedures

Abaqus/CAE can perform remeshing with the following Abaqus/Standard procedures:

- *Static Stress Analysis* (general and linear perturbation).
- *Quasi-Static Analysis*.
- *Uncoupled Heat Transfer Analysis*.
- *Fully Coupled Thermal-Stress Analysis*.
- *Coupled Thermal-Electrical Analysis*.
- *Coupled Pore Fluid Diffusion and Stress Analysis*.

Selection of Error Indicators Influencing Adaptive Remeshing

Products: Abaqus/Standard Abaqus/CAE

References:

- *Error Indicator Output*
- *About Adaptive Remeshing*
- *Abaqus/Standard Output Variable Identifiers*
- **CONTACT OUTPUT*
- **ELEMENT OUTPUT*
- *Understanding adaptive remeshing*
- *Controlling adaptive remeshing*

Overview

Your selection of which error indicator variables to use in adaptive remeshing rules for a particular analysis should take into consideration:

- characteristics of the error indicator variables;
- which fields exist and are of interest; and
- the nature of the loading.

Error Indicator Characteristics

Error indicator output variables provide estimates of solution accuracy (see *Error Indicator Output*). In the context of adaptive remeshing, error indicators help determine where the mesh should be refined or coarsened to achieve the specified accuracy targets (see *About Adaptive Remeshing* and *Solution-Based Mesh Sizing*). This Section discusses additional characteristics of error indicators in the context of how well-suited they are for influencing adaptive remeshing in various analysis types.

Which Fields Exist and Are of Interest

Certain variables apply naturally to certain types of analyses. For example, the heat flux indicator (HFLERI) is used in analyses with temperature degrees of freedom. When selecting error indicator variables in the **Remeshing Rule** editor in Abaqus/CAE (see *What are remeshing rules?*), your choices will be restricted to variables available for the selected procedure type.

The Nature of the Loading

Some error indicator variables only indicate discretization error at the current analysis time—the particular increment in a step. Other error indicator variables provide a record of the solution history up to the current analysis time. For example, if your simulation involves non-proportional loading or a significantly nonlinear response, you will typically see better adaptive remeshing results when using error indicator variables that record the solution history. *Table 1* lists the error indicator variables applicable to adaptive remeshing and indicates whether they record the solution history.

Table 1: Error indicator variables applicable to adaptive remeshing that record the solution history.

Solution Quantity	Error indicator variable (c_e)	Records the solution history?
Element energy density	ENDENERI	Yes
von Mises stress	MISESERI	No
Equivalent plastic strain	PEEQERI	Yes
Plastic strain	PEERI	No
Creep strain	CEERI	No
Heat flux	HFLERI	No
Electric flux	EFLERI	No
Electric potential gradient	EPGERI	No

By default, when you create a remeshing rule, error indicators are specified for the final increment of the final step of your analysis and adaptive remeshing is based on error indicators in this final increment. When you select an error indicator that records the solution history, this default error indicator specification is appropriate for almost all analyses. However, for other error indicator variables that do not record the solution history, you may find it appropriate (for multistep cases with non-proportional loading, for example) to define mutiple remeshing rules for the same region, with each rule applied to a different step.

The examples that follow provide simple illustrations of typical cases and show appropriate choices of error indicator output variables.

Linear Response Example

Figure 1 illustrates the simplest load case, where the load is proportional to the step time and the model's response is linear. In this case the solution at the final increment would be proportional to any other increment. Therefore, it is appropriate to base the remeshing on the value of the error indicator in the last increment for any choice of error indicator variable.

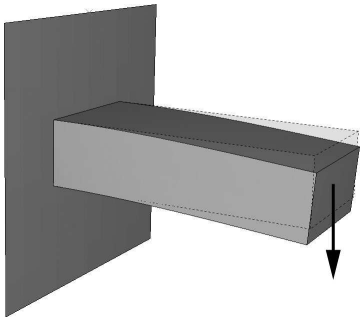


Figure 1: Proportional-loading, linear-response example: small deflection of a cantilever.

Monotonic Response Example

Figure 2 illustrates a more general case, where the model has a nonlinear response—in this case resulting from a geometric nonlinearity—and the loading is monotonic but not generally proportional to the step time.

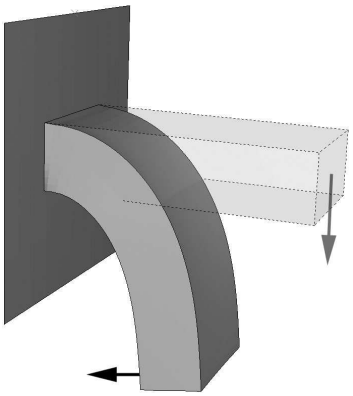


Figure 2: Monotonic response example: large deflection of a cantilever.

The response of the model is slightly more general because the solution at a particular increment is not proportional to the solution at the final increment. However, the value of the error indicator output in the final increment still reflects the extreme of the model's response to the load history. Therefore, it is appropriate to base the remeshing on the value of the error indicator in the last increment for any choice of error indicator variable.

General Response Example

Figure 3 illustrates a case where the loading characteristics change dramatically during the analysis.

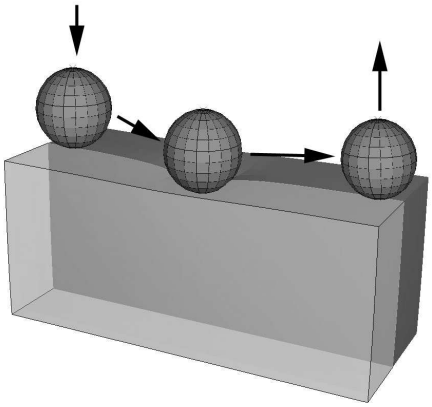


Figure 3: General response example: block subjected to a rigid indenter.

Your choice of error indicator in this case will depend on the material model. The element energy density error indicator, ENDENERI, will account for the complexity of load history (and lead to an adapted mesh that provides an accurate solution through the analysis) regardless of the material type. If plastic deformation occurs, you also have the option to use the equivalent plastic strain, PEEQERI, or plastic strain, PEERI, error indicators. Plastic strain and the plastic strain error indicator generally do not capture history effects; for example, they do not account for peak straining in models undergoing symmetric strain reversals. This example, however, involves no strain reversals; therefore, PEERI would be a valid error indicator choice.

General Multistep Response Example: Die Forming and Springback

Figure 4 illustrates a further generalization of a general response. Here, a forming operation is simulated, and different steps are used for different stages of the operation.

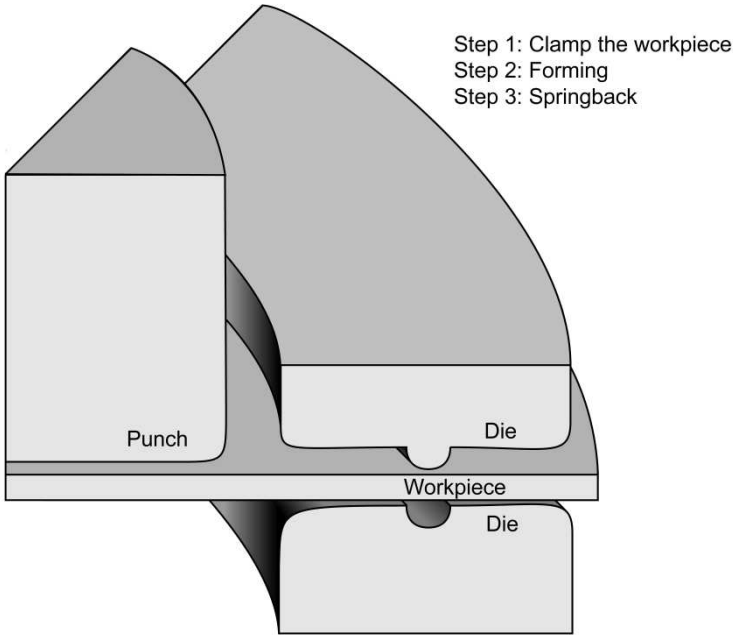


Figure 4: General multistep response example.

In this case the response of the model varies from step to step. You will typically want the error indicator to capture the extreme of the model's response to the load history adequately. However, you do not know if any particular increment captures this extreme. Therefore, you should select an error indicator variable that records the solution history.

Solution-Based Mesh Sizing

Products: Abaqus/Standard Abaqus/CAE

References:

- *About Adaptive Remeshing*
- *Selection of Error Indicators Influencing Adaptive Remeshing*
- *Understanding ALE adaptive meshing*
- *Advanced meshing techniques*

Overview

Solution-based mesh sizing:

- is performed in Abaqus/CAE; and
- operates on error indicator output variables and your remeshing rule parameters (see *Creating a remeshing rule*) to determine an improved element size distribution for your mesh.

Basic Operation of the Sizing Method

The sizing method calculates new element sizes during the adaptive remeshing process. Abaqus/CAE applies the sizing method to a field of error indicator variables and their corresponding base solution variables over the region defined by the remeshing rule. The output of a sizing method is a set of scalar sizes located at the nodes in the region defined by the remeshing rule. *Figure 1* illustrates the sizing operation. *Figure 1* shows the base solution and error indicator distributions after the first remesh iteration. The sizing method determines that the element size should be reduced in the region of greatest error indicator and increased in the region of the lowest error indicator. The mesh that is generated from these target element sizes is illustrated.

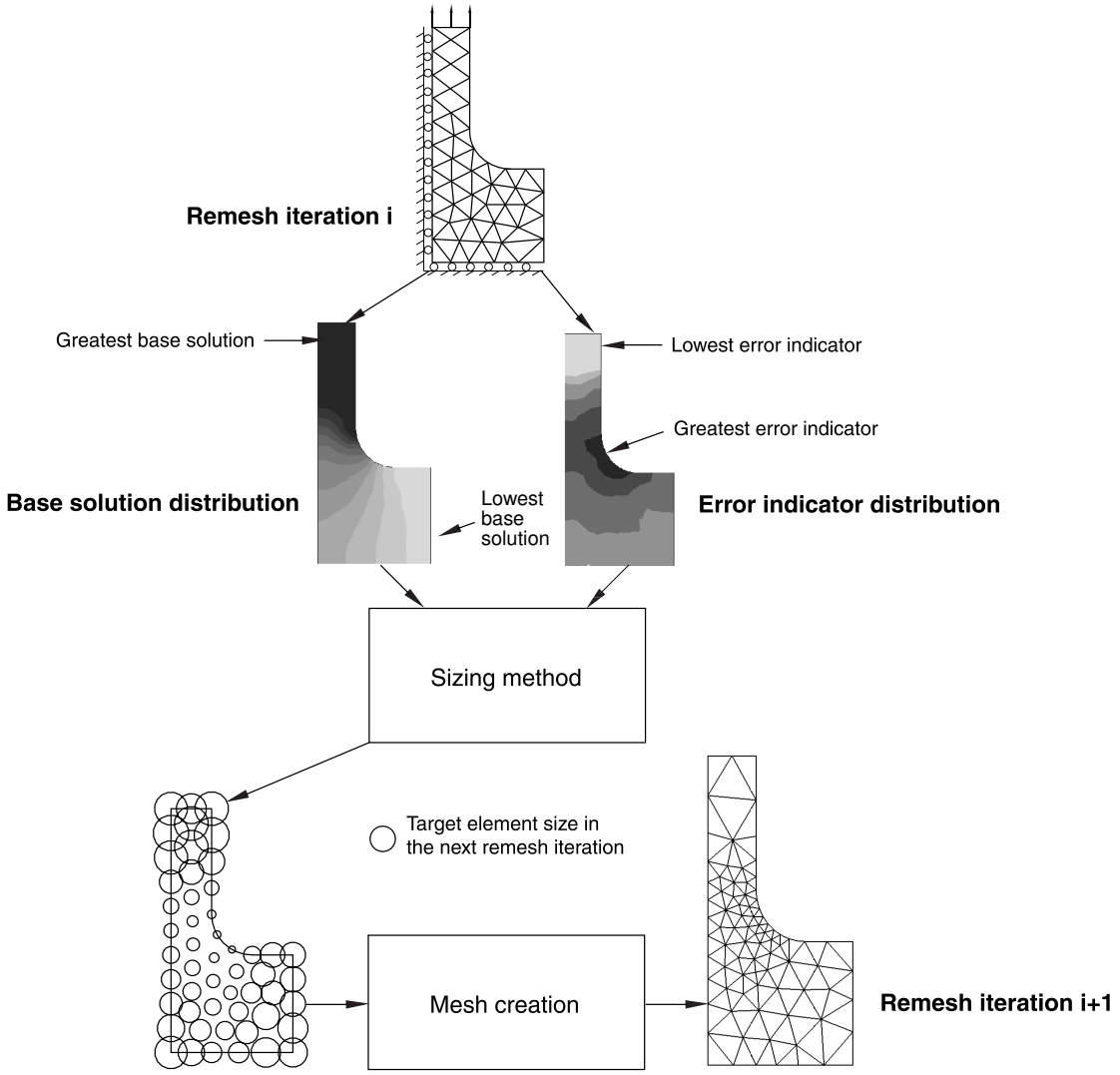


Figure 1: Sizing method operation and interaction with meshing.

Characteristics of Error Indicators

The sizing method and parameter settings that you select have a significant impact on how adaptive remeshing changes the error indicator distribution in your model. You may, for example, choose a sizing method that aggressively reduces error indicators only near a stress riser. In other cases, where the global response of your structure is more important than local effects, you may choose a sizing method that attempts to reduce the error indicators to a uniform level throughout the region. To understand how the sizing methods affect the error indicators, you should first understand typical characteristics of the error indicator variables.

Figure 2 provides an illustration of an error indicator and corresponding base solution distribution on a generalized slice through a model.

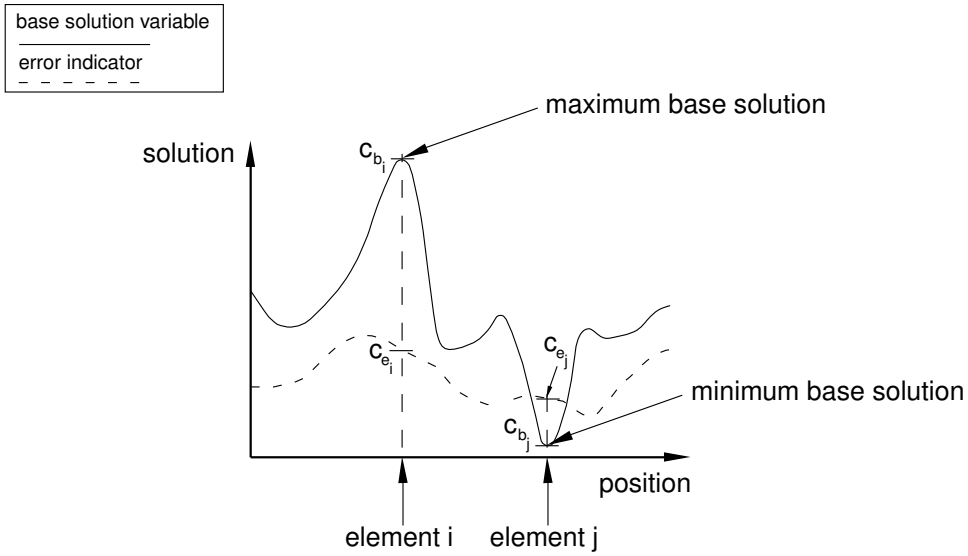


Figure 2: Error indicator and base solution distribution.

Figure 2 illustrates the following error indicator characteristics:

- In regions where the value of the base solution is high, such as for element “i” in *Figure 2*, error indicator values can be low relative to local values of the base solution. In many cases you may want to use mesh refinement to drive these error indicators even lower.
- In regions where the base solution is low, such as for element “j” in *Figure 2*, error indicator values can be high relative to the local values of the base solution. In many cases you may not be interested in obtaining an accurate solution in these regions.

These characteristics can affect your decision on which sizing method to use and what parameters to set in the sizing method.

Sizing Methods

Sizing methods employ the concept of an error target, η, which is expressed in a normalized percentage form and which defines a general goal

$$\left(\frac{\mathbf{c_e}}{\mathbf{c_b}}\right) \times 100 \Rightarrow \eta,$$

where $\mathbf{c_e}$ is a measure of the error indicator and $\mathbf{c_b}$ is a measure of the base solution. Based on your definition of the error targets when you created the remeshing rule, Abaqus/CAE creates a size distribution that attempts to meet your error target in the subsequent analysis job using the remeshed model. The specific meaning of an error target depends on your choice of the sizing method.

Abaqus/CAE provides two fundamental sizing methods: **Minimum/maximum control** and **Uniform error distribution**. You can also choose a third method, **Default method and parameters**, which results in Abaqus/CAE choosing one of the fundamental sizing methods for you, based on your choice of error indicator variable.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**

Minimum/Maximum Control

The minimum/maximum control method provides the most flexibility in the remeshing of your model. This method has the following characteristics:

- Two error indicator targets for controlling the sizing. η_{max} controls the sizing in regions where the base solution (such as stress) is highest, and η_{min} controls the sizing in regions where the base solution is lowest.
- A continuous variation in error targets between regions of high and low base solution values, with a bias factor parameter provided to control the variation.
- To avoid excessive refinement at elements with a small base solution, a global averaged element base is chosen when the element base solution is smaller than the global averaged element base.
- If singularities are present in the remeshing rule region, this method will fail to satisfy the error target because the maximum base solution, which occurs at the location of the singularity, is unbounded.

You can either allow Abaqus to choose the targets automatically, or you can specify the error targets. Similarly, you can accept the default bias factor displayed by Abaqus/CAE, or you can specify a qualitative bias factor.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Minimum/Maximum control**

Allowing Abaqus/CAE to Choose the Error Targets

If you specify the minimum/maximum error control method without setting error targets, Abaqus/CAE automatically chooses the error targets, η_{max} and η_{min}. Both targets are computed as a fraction of the error indicator result in the previous remesh iteration analysis. Automatic error target reduction is a good choice for mesh refinement studies, where you have no specific error target goal but want to see the impact of mesh refinement on your results.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Automatic error target reduction**

Specifying the Error Targets

As an alternative to automatic error target reduction, you can specify the two error targets, η_{max} and η_{min}. *Figure 2* illustrates these two locations. η_{max} is applied to element i, and η_{min} is applied to element j.

Using the value of the two error targets, Abaqus/CAE applies a sizing method that attempts to meet both η_{min} and η_{max} at their respective locations.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Fixed error targets**; enter the maximum base solution error indicator target, η_{max}, and the minimum base solution error indicator target, η_{min}

Bias Factor

You can use the bias factor definition in the remeshing rule to further tune the distribution of sizing between maximum and minimum base solution locations. The bias factor defines the gradient of the size distribution between these two extremes in your remesh region, as shown in *Figure 3*.

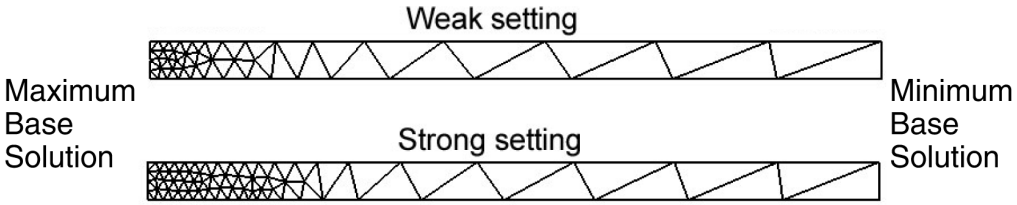


Figure 3: The impact of the bias factor on the element size distribution.

You can set this factor between two qualitative extremes, “weak” and “strong.” At the weak extreme, element sizes will increase most quickly at locations moving away from the maximum base solution. At the strong extreme, element sizes will increase most slowly. The default setting is a bias toward the strong extreme.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Mesh Bias**; drag the slider to a setting between **Weak** and **Strong**

Uniform Error Distribution

The uniform error distribution method provides a single error indicator target, η, for controlling the sizing. Abaqus/CAE applies a sizing method such that the total error in the remeshing rule region is distributed uniformly across all the elements and satisfies the given error indicator target. This method attempts to satisfy the error indicator target collectively for the whole remeshing rule region but not at every element. Therefore, the presence of singularities will not prevent the adaptivity process from achieving the error target.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Uniform error distribution**

Allowing Abaqus/CAE to Choose the Error Target

If you specify the uniform error distribution method without setting an error target, Abaqus/CAE automatically chooses the error target, η. The target is computed as a fraction of the error indicator result in the previous remesh iteration analysis. Automatic error target reduction is a good choice for mesh refinement studies, where you have no specific error target goal but want to see the impact of mesh refinement on your results.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Automatic error target reduction**

Specifying the Error Target

As an alternative to the automatic error target reduction, you can specify the single error target, η. When you use the uniform error distribution method, Abaqus/CAE compares the error target to a global norm of a normalized form of the error indicator. Such an approach ensures a globally converging mesh within the region.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**: choose **Fixed error target**; enter the error indicator target, η

Default Sizing Methods and Parameters

This method results in application of the **Automatic error target reduction** form of either the **Minimum/maximum control** or **Uniform error distribution** method, with the method applied based on the error indicator variable according to *Table 1*.

Table 1: Default sizing method for each error indicator.

Solution Quantity	Error indicator variable	Default sizing method
Element energy density	ENDENERI	Uniform error distribution
von Mises stress	MISESERI	Minimum/maximum control
Equivalent plastic strain	PEEQERI	Minimum/maximum control
Plastic strain	PEERI	Minimum/maximum control
Creep strain	CEERI	Minimum/maximum control
Heat flux	HFLERI	Uniform error distribution
Electric flux	EFLERI	Minimum/maximum control
Electric potential gradient	EPGERI	Minimum/maximum control

When your remeshing rule refers to multiple error indicators, sizing methods will be applied independently to each error indicator variable with the resulting local size based on the smallest size calculated from each sizing method.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Default methods and parameters**

Example: Plate with a Circular Stress Riser

The difference between the basic behavior of the minimum/maximum control and the uniform error distribution methods is illustrated by a simple example. *Figure 4* shows the stress result for a simple loading of a plate with a hole.

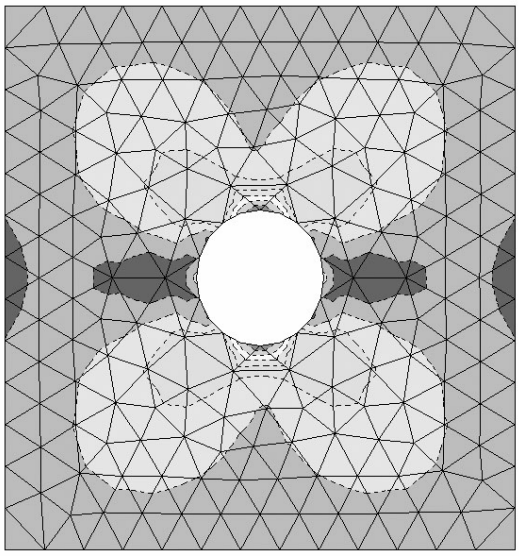


Figure 4: Initial mesh and von Mises stress distribution for a plane stress plate with a hole, subjected to a uniform horizontal boundary traction.

Minimum/Maximum Control

Figure 5 illustrates the adaptive mesh that was generated by Abaqus/CAE when the user selected the minimum/maximum control method and specified the two error targets ($\eta_{\min}$ and $\eta_{\max}$). In this example $\eta_{\min}$=5%

and η_{max}=1%, and the mesh bias is set to the default setting. These settings result in a mesh that focuses tightly around the hole, the stress riser, while transitioning smoothly to a relatively coarse mesh away from the hole.

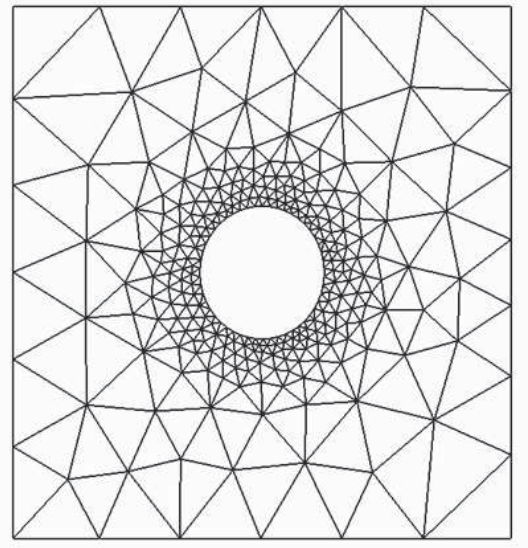


Figure 5: Adaptive remesh resulting from the minimum/maximum control sizing method.

Uniform Error Distribution

Figure 6 illustrates the adaptive mesh that was generated by Abaqus/CAE when the user selected the uniform error distribution method and specified the single uniform error indicator target (η). In this example η=1%. This setting results in a mesh that focuses around the hole, the stress riser, while also refining the mesh in less stressed regions.

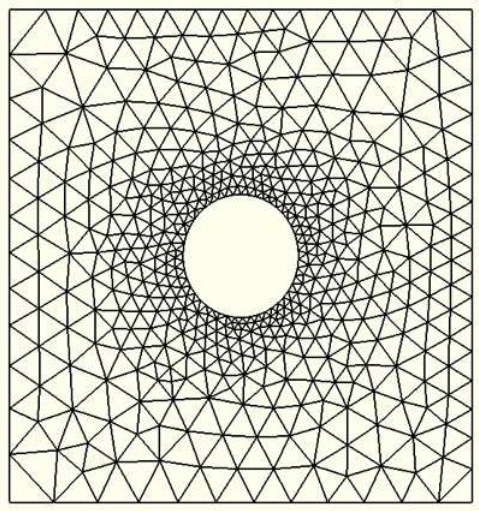


Figure 6: Adaptive remesh resulting from the uniform error distribution sizing method.

Impact of Additional Remeshing Rule Settings

You specify the sizing method when you create a remeshing rule, and the sizing method calculates new element sizes during the adaptive remeshing process. However, the following additional settings in the remeshing rule can affect the mesh generated by Abaqus/CAE, regardless of the sizing method that you selected:

- region selection,
- step and frame selection,
- size constraints,
- approximate maximum number of elements, and
- refinement and coarsening rate factors.

Region Selection

Sizing methods are defined across sets of elements, corresponding to the regions over which the remeshing rules were applied in Abaqus/CAE. Within each set of elements, Abaqus/CAE applies the sizing operation to the error indicator variables specified in the remeshing rule. The results of the sizing operation are based on the extrapolation of whole element calculations to the nearest nodes, and the results are node based.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Edit Region**

Step and Frame Selection

Abaqus applies sizing operations to error indicator variables from only the last available frame in a specified step. See *Error Indicator Characteristics* for a discussion of how your selection of the step, frame, and error indicator can affect your ability to capture the response in transient analyses.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Step and Indicator**: **Step**; select the step to which the rule is applied

and

Mesh module: **Create Remeshing Rule**: **Step and Indicator**: **Output Frequency**; choose either **Last increment of step** or **All increments of step**

Size Constraints

When you create the remeshing rule, you can constrain the sizing operation from specifying elements greater than or less than size constraints that you define for the remesh rule region. Abaqus/CAE provides default settings for these constraints.

- The default minimum element size constraint is 1% of the default boundary seed size for the part instance to which the remeshing rule is applied.
- The default maximum element size constraint is 10 times the default boundary seed size for the part instance to which the remeshing rule is applied.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Element Size**

Approximate Maximum Number of Elements

For a remeshing rule you can specify an approximate limit for the maximum number of elements. By using this constraint, you can control the cost of your analysis and ensure that unreasonably large meshes are not created. If the target error requires more elements than the specified limit when this constraint is defined, Abaqus/CAE will reduce the target error internally so that the generated elements approximately satisfy the specified element count. The use of this constraint may prevent an adaptivity process from achieving the error targets. By default, this constraint is not active.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Approximate maximum number of elements**

Refinement and Coarsening Rate Factors

The refinement and coarsening factors that you specify define a constraint on the mesh size in terms of iteration to iteration changes to the mesh. These factors modulate the aggressivity of the sizing methods. The refinement factor controls the refinement of the mesh or the introduction of smaller elements. The coarsening factor controls the coarsening of the mesh or the introduction of larger elements. Abaqus/CAE provides default settings for these rate factors, which are designed to prevent excessive coarsening or prohibitively expensive refinement in a single remesh iteration.

The refinement factor can have a significant effect on the convergence of the adaptive meshing procedure. Once you have settled on sizing method parameters that work well for your application, you may be able to achieve faster and more efficient mesh convergence by increasing the refinement factor. In cases where your adaptivity process is not converging well, however, an increased refinement factor could result in an excessive increase in elements in a remesh iteration.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Rate Limits**

Reconciling Overlapping Remeshing Rules

Abaqus/CAE imposes no restrictions on the region or the steps associated with your remeshing rules. You can apply multiple remeshing rules and, hence, sizing functions to the same region at the same time. Similarly, you can specify remeshing rules that overlap one another. When Abaqus/CAE generates the new mesh, it considers all of the remeshing rules at all of the locations and uses the smallest calculated element size to drive the meshing algorithm.

Mesh-to-Mesh Solution Mapping

Products: Abaqus/Standard

References:

- **MAP SOLUTION*

Overview

Mapping a solution from one mesh to another is a step in a remeshing analysis technique, where a mesh that has deformed significantly from its original configuration is replaced by a mesh of better quality and the analysis continues. The solution mapping technique:

- is used when elements become so severely distorted during an analysis that they no longer provide a good discretization of the problem;
- maps the solution from an old, deformed mesh to a new mesh so that the analysis can continue; and
- can be used only with continuum elements.

Refer to *About Adaptivity Techniques* for a high-level discussion comparing this and other Abaqus adaptivity methods.

When to Remesh

Abaqus/Standard uses a Lagrangian formulation: the mesh is attached to the material and, thus, deforms with the material. When the strains become large in geometrically nonlinear analyses, the elements may become so severely distorted that they no longer provide a good discretization of the problem. Severe distortion may occur in rubber elasticity problems or in plastic or viscoplastic calculations, especially when modeling manufacturing processes. When severe distortion occurs, it is necessary to remesh: to create a new mesh better designed to continue the analysis and to map the old-model solution onto this mesh.

You must decide when remeshing is needed. This decision can be assisted by looking at the magnitude of strains that have occurred during the phase of the analysis using a particular mesh, as discussed later. When remeshing is required, a new mesh for the deformed object must be generated using the mesh generation capability in Abaqus or an external mesh generator. The analysis is then continued as a new problem using the new mesh. In most cases it will be desirable to transfer the solution from the old mesh to the new mesh.

Discontinuity in the Solution

Whenever the solution is mapped from another mesh, you can expect that there will be some discontinuity in the solution because of the change in the mesh and as a consequence of the solution mapping algorithm. If the discontinuity is significant, it is an indication that the meshes are too coarse or that the remeshing should have been done at an earlier stage before too much distortion occurred.

The remeshing technique works well, provided that the meshes are sufficiently fine for the problem and that the remeshing is done before the elements become too distorted.

Remeshing Criterion

The first requirement for remeshing is some indication that the mesh is becoming distorted in regions where this distortion could cause the solution to be inaccurate. One possible criterion for remeshing would be extreme element distortion in areas where high strain gradients need to be resolved accurately. Inaccuracy is less of a

concern if the distorted elements have moved into an area where further changes in the strain field are uniform; the elements can represent states of constant strain accurately no matter how distorted they are. Ultimately, however, the decision to remesh is a matter of judgment.

Generating a New Mesh

Once you have decided that the current mesh is inadequate, a new mesh that is more suitable to the current state of the problem must be generated by using the mesh generation capabilities in Abaqus or an external mesh generator. Deformed configuration plots may be useful to provide data about the current shape of the object being modeled. Usually the external surface can be defined for use in a mesh generator from the results file output at the sets of nodes that form the surfaces of the body. See *Erosion of material (sand production) in an oil wellbore* and *Upsetting of a cylindrical billet: quasi-static analysis with mesh-to-mesh solution mapping (Abaqus/Standard) and adaptive meshing (Abaqus/Explicit)*.

Remeshing a Contact Problem

In a region of contact the new mesh must conform closely to the shape of the surface from the old analysis. This requirement is especially important for problems involving contact between two deformable bodies; if the surfaces defined by the new mesh are even slightly different from the surfaces in the old analysis, the contact algorithms may fail to converge.

Specifying the Solution to Be Interpolated onto the New Mesh

The simulation is continued by interpolating the solution onto the new mesh from the output databases generated with the old mesh.

Specifying the Time at Which the Solution Must Be Read

Solution transfer will occur, by default, from the latest step and increment for which solution variables are available. Alternatively, you can specify the step and increment at which the old solution will be read.

Input File Usage: **MAP SOLUTION*, STEP=*step*, INC=*increment*

Obtaining Equilibrium

An initial step should be included to allow Abaqus/Standard to check for equilibrium after this interpolation has been done. By default, Abaqus/Standard resolves the stress unbalance linearly over the step (see the discussion on establishing equilibrium when an initial stress field is applied in *Initial Conditions*). You can choose to have the stress unbalance resolved in the first increment instead.

Input File Usage: Use the following option to have Abaqus/Standard resolve the stress unbalance linearly over the step:

**MAP SOLUTION*, UNBALANCED STRESS=RAMP

Use the following option to have Abaqus/Standard resolve the stress unbalance in the first increment of the step:

**MAP SOLUTION*, UNBALANCED STRESS=STEP

Translating and Rotating the Old-Job Mesh

The mesh from the old job can be repositioned prior to performing the mapping by giving a translation and/or rotation relative to the global origin. Specify a translation by giving a translation vector. Specify a rotation by giving two points to define a rotation axis plus a right-handed angular rotation around that axis.

Input File Usage: **MAP SOLUTION*, STEP=*step*, INC=*increment*
translation vector data
rotation axis and angular rotation data

Required Output from the Old Job

The files required for restart and the output database must be requested for the old job. Nodal displacement results are not output automatically from the old job; you must explicitly request output of the displacement variable U for all nodes, as described in *Writing Nodal Output to the Output Database*. Alternatively, you can request preselected field output and obtain node displacement output sufficient for solution mapping.

In fully coupled procedures you must request nodal output of the coupled field variable to the output database (see *Table 1*).

Table 1: Output database nodal output requirements for fully coupled procedures.

Procedure	Nodal output variable
Fully Coupled Thermal-Stress Analysis	NT11
Coupled Pore Fluid Diffusion and Stress Analysis	POR
Geostatic Stress State	POR

Identifying the Old Job

Specify the name of the old job from which restart and results data will be obtained by using the **oldjob** parameter in the command for running Abaqus or by answering a request made by the command procedure (see *Abaqus/Standard and Abaqus/Explicit Execution*). The files required from the old job include: the restart file (`.res`), the output database (`.odb`), the model database (`.mdl`), the state database (`.stt`), and the part (`.prt`) file.

Solution Mapping Algorithm

Solution mapping operates by interpolating results from nodes in the old mesh to points (either nodes or integration points) in the new mesh. The first step, therefore, involves associating solution variables with nodes in the old mesh. For nodal solution variables, such as nodal temperature or pore pressure, the association is already made. For integration point variables Abaqus obtains the solution variables at the nodes of the old mesh by extrapolating values from the integration points to the nodes of each element and then averaging these values over all similar elements abutting each node.

Next, the location of each point in the new mesh is obtained with respect to the old mesh. The new mesh points include integration points in all cases and nodes in procedures that record nodal state in addition to displacements (for example, nodal temperatures in coupled temperature-displacement procedures).

1. The element (in the old mesh) in which the point lies is found, and the point's location in that element is obtained. (This procedure assumes that all points in the new mesh lie within the bounds of the old mesh: warning messages are issued if this is not so, and the values of the variables are set to zero.)
2. The variables are then interpolated from the nodes of the old element to the points in the new model.

All necessary variables are interpolated automatically in this way so that the solution can proceed with the new mesh.

Solution Diffusion

This algorithm introduces some diffusion in the mapped solution. The effect of the diffusion scales with the solution gradient in the old mesh; hence, even for regions of the model where the mesh does not change from the old to the new model, diffusion due to the mapping can result in significantly different mapped quantities when the old-mesh solution gradient is high. You can moderate this effect by refining the old mesh in regions where solution gradients are high or by remeshing earlier.

Procedures

The solution mapping capability is supported for the following procedures:

- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*
- *Geostatic Stress State*

Initial Conditions

The solution mapped from the initial analysis forms the initial conditions for the remeshed analysis. Initial conditions such as temperature for a pure stress/displacement analysis can be specified. Any other specified initial conditions will be ignored.

Boundary Conditions

Boundary conditions are not carried over from the old mesh to the new mesh. The boundary conditions applied at the beginning of the remeshed analysis should normally be the same as those in effect at the step and increment selected from the initial analysis. Although boundary conditions can be changed, the problem may fail to converge if the structure is far from an equilibrium state.

There are no restrictions on applying boundary conditions in a mapped solution analysis. Boundary conditions can be applied to all available degrees of freedom in the same way as they are applied in an analysis without a mapped solution (see *Boundary Conditions*).

Loads

There are no restrictions on applying loads in a mapped solution analysis. Loads can be applied in the same way as they are applied in an analysis without a mapped solution.

The loads applied at the beginning of the remeshed analysis should normally be the same as those in effect at the end of the initial analysis. Although the loads can be changed, the problem may fail to converge if the structure is far from an equilibrium state.

Predefined Fields

Temperature and field variables are mapped from the old mesh to the new mesh. If the number of field variables is changed in the remeshed analysis, the number common to both analyses will be transferred. Predefined fields

can be modified in the same way as they are modified in an analysis without solution mapping (see *Predefined Fields*).

Material Options

Except for the anisotropic hyperelastic model, any of the mechanical constitutive models available in Abaqus can be used in a mapped solution analysis (see *Abaqus Materials Guide*). There is no restriction on agreement between material models in the old and new analyses. The solution mapping algorithm will transfer those variables common to both models. You must ensure that the material models are compatible.

Elements

The solution mapping capability can be used only with continuum elements (see *Solid (Continuum) Elements*) excluding elements C3D10MP and C3D10HS.

Output

There is no output specific to a mapped solution analysis. Output can be requested in the same way as in an analysis without a mapped solution. The output variables available in Abaqus are listed in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

```
*HEADING
*NODE
Data lines to define the new-model nodes occupying the space of the old model
in its deformed configuration
*ELEMENT
Data lines to define the new-model elements occupying the space of the old model
in its deformed configuration
…
*MAP SOLUTION, STEP=step, INC=inc
translation and rotation data
*STEP
*STATIC (or *COUPLED TEMPERATURE-DISPLACEMENT or *GEOSTATIC
or *SOILS or *VISCO)
…
*END STEP
```

Fatigue Crack Propagation with Adaptive Remeshing

Products: Abaqus/Standard

References:

- *Linear Elastic Fatigue Crack Growth Analysis*
- *Contour Integral Evaluation*
- *Linear Elastic Fatigue Crack Growth Behaviors*
- **ADAPTIVE REMESH*
- **ADAPTIVE REMESH RETAIN*
- **CONTOUR INTEGRAL*
- **FATIGUE*
- **FRACTURE CRITERION*

Overview

The workflow for fatigue crack propagation with adaptive remeshing is based on well-established Abaqus/Standard fracture/fatigue technology in conjunction with tetrahedral element–based adaptive remeshing technology.

The workflow assumes a preexisting seam crack in the structure. It is semi-automated and involves automatic execution of a series of successive Abaqus/Standard analyses with progressively longer crack lengths, without requiring any direct user intervention. Each Abaqus/Standard analysis subjects the cracked structure to a complete load cycle. The workflow makes use of extrapolation techniques, which removes the need to model every single load cycle.

You can use the capability only for linear elastic fatigue crack growth analysis; state mapping between any two successive Abaqus/Standard analyses is not necessary.

You can use Abaqus/CAE to create the initial model with the preexisting crack (seam). However, you must use an Abaqus/Standard input file to define the fatigue crack propagation with adaptive remeshing workflow. For more information, see *Creating a seam*.

Overview

The general workflow for performing a fatigue crack growth analysis with adaptive remeshing is as follows:

1. Prepare the model.
   a. Create a model containing a single seam crack.
   b. Partition the overall domain into adaptive and (optionally) nonadaptive subdomains.

      The adaptive domain must contain the seam crack. The mesh in the adaptive remeshing is updated automatically following each Abaqus/Standard analysis.

   c. Mesh the adaptive subdomain with second-order tetrahedral (C3D10) elements with a refined, but otherwise unstructured, mesh around the crack front.

      The optional nonadaptive subdomain can include any of the stress/displacement elements available in the Abaqus/Standard element library.

   d. Define a surface-based tie constraint between the adaptive and the nonadaptive subdomains.
   e. Ensure that the model (Abaqus input file) does not contain any parts and assemblies.

If you create the model in Abaqus/CAE, write out an Abaqus/Standard input file without parts and assemblies.

2. Launch the fatigue crack growth with adaptive remeshing workflow.
3. Abaqus/Standard computes the new crack front by performing a series of fatigue calculations that are based on the current crack front. Each set of calculations constitutes an increment of crack growth.
   a. For a given crack length, Abaqus/Standard computes the energy release rates or the stress intensity factors at each node along the crack front.
   b. Using these quantities Abaqus/Standard calculates the updated (due to crack growth) locations of the crack front nodes and the number of load cycles required to reach this state from the beginning of the analysis.
   c. After each increment of crack growth, Abaqus/Standard uses the tetrahedral element–based adaptive remeshing technology to create a new mesh that accommodates the updated crack front.
   d. It also automatically creates a new Abaqus/Standard model with an updated ("longer") crack.
4. Abaqus/Standard then uses this new model to continue the fatigue crack growth calculations by essentially repeating Step 3 above.
5. This process continues until either:
   - the maximum number of cycles that you specified is reached,
   - the structural component has completely lost its load carrying capacity, or
   - the crack front reaches the adaptive domain boundary (assuming both adaptive and nonadaptive domains exist).

Figure 1 shows the initial mesh (left) of a structure with a crack, as well as a new mesh (right) of the same structure with a longer crack. The new mesh is more refined around the current crack front, while it is less refined in the wake of the crack.

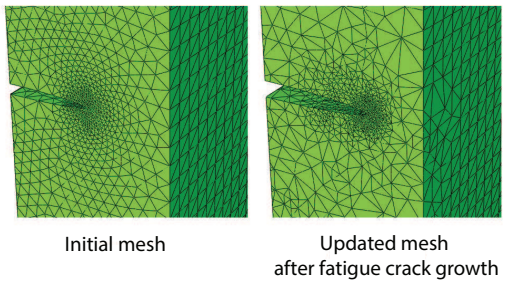


Figure 1: Use of adaptive remeshing to generate a new mesh that conforms to the updated crack front as the crack propagates.

Figure 2 outlines the different phases of the workflow.

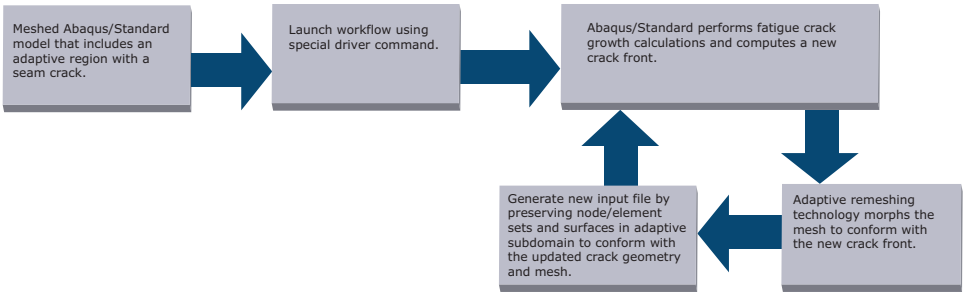


Figure 2: Phases of the fatigue crack growth analysis with adaptive remeshing workflow.

Preparing Your Model for Adaptive Remeshing

You must define the initial model using the Abaqus/Standard input file interface (see *Input Syntax Rules*), and the model must not contain any parts and assemblies. You can use Abaqus/CAE to create the model and write an input file for the model.

In general, the model should consist of two subdomains:

- an adaptive subdomain containing the crack, and
- a nonadaptive subdomain for the rest of the model.

The initial mesh in the adaptive domain must contain a refined mesh with several layers of second-order tetrahedral elements around the crack front. You must ensure that the mesh along the crack front is refined sufficiently to capture the large gradients expected in the vicinity of a sharp crack front. Abaqus/Standard uses the averaged element length along the initial crack front to define the seeds for the local mesh refinements around the updated crack front during each adaptive remeshing.

Activating the Adaptive Remeshing Capability and Identifying the Adaptive Subdomain in a Model

A typical model for this workflow consists of an adaptive and (optional) nonadaptive subdomains. To illustrate the approach for selecting the subdomains, refer to the example below that represents a quarter model of a thermal ring (*Figure 3*). This model consists of an inner hub containing a crack that grows during the analysis and an outer rim that either does not have any crack or has a stationary crack (that is, a crack that will not grow during the analysis). In this example, you should define the inner hub as the adaptive subdomain, and you should treat the outer rim as a nonadaptive subdomain.

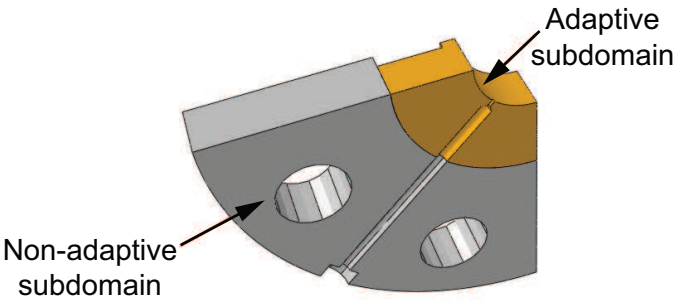


Figure 3: A model containing an adaptive subdomain and a nonadaptive subdomain.

You must identify the adaptive subdomain in the model by associating the adaptive remesh definition with the appropriate element set.

Input File Usage: Use the following option to identify the adaptive domain in the model:

*ADAPTIVE REMESH, ELSET=*element set name*

Retaining Mesh/Mesh Geometric Entities for Load and Boundary Condition Considerations

During adaptive remeshing, Abaqus/Standard needs to preserve the element sets, node sets, and surfaces that are part of the adaptive subdomain and are involved in the application of loads, boundary conditions, constraints, nodal transformation, section definitions, etc. To enable this, you must explicitly specify the entities (sets and surfaces) that must be preserved or retained. Following each instance of adaptive remeshing, Abaqus/Standard updates the members of these sets and/or surfaces and recreates the relevant Abaqus/Standard options and the corresponding data in the new model. In the (optional) nonadaptive subdomain, all options remain unchanged in the updated models following each instance of remeshing.

Tracking the Sets and Surfaces in the Adaptive Domain

To illustrate the requirements for preserving mesh and geometric entities, such as sets and surfaces, refer to the examples below. The first example models a semi-elliptical crack in a rig subjected to internal pressure loading (see *Figure 4*). The second example models a single-edge crack panel, which requires three separate section definitions, as shown in *Figure 5*. The numeric labels superposed on the figures below summarize the mesh/mesh geometric entities that need to be preserved.

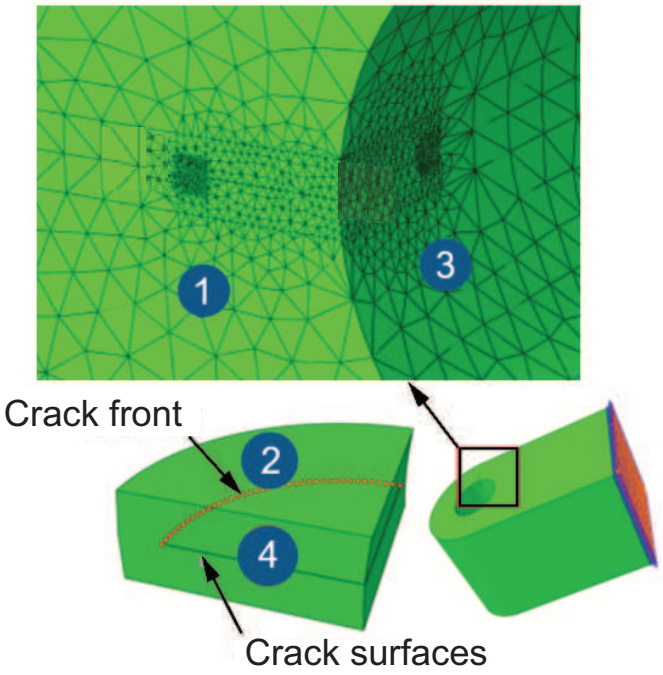


Figure 4: A semi-elliptical crack in a rig subjected to internal pressure.

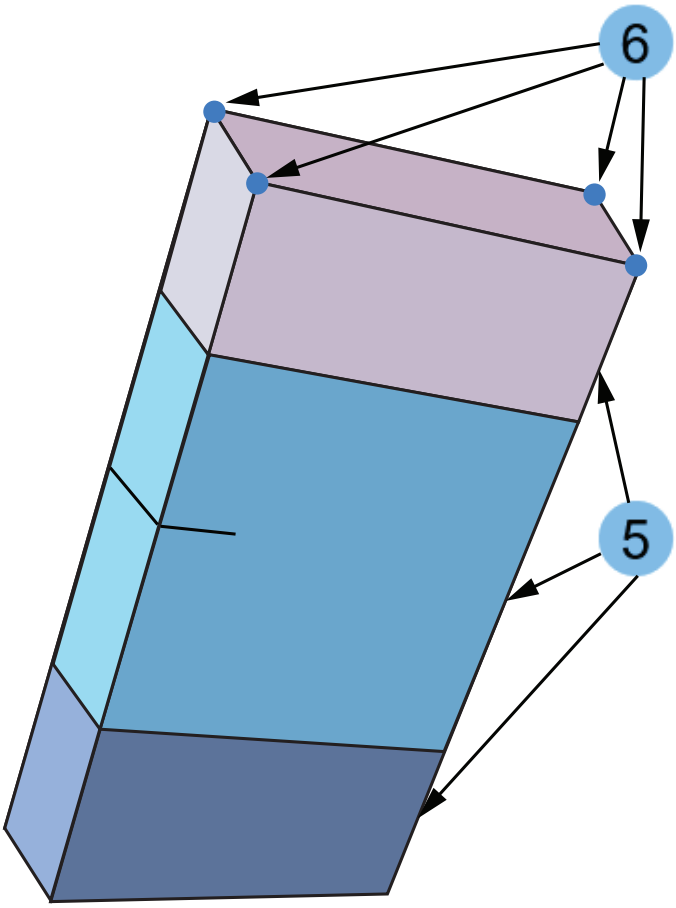


Figure 5: A single edge crack panel with multiple section definitions

1. Updated node sets for applying (symmetry) boundary conditions
2. Updated node sets along the crack front for contour integral calculations
3. Updated element facets for applying the internal pressure loading
4. Updated element facets on the crack faces for enforcing contact constraints
5. Updated element set for each of the section definitions
6. Updated node sets with discrete nodes for applying boundary conditions

Specifying the Retained Sets and Surfaces in the Adaptive Subdomain of the Model

Based on the requirements outlined above, you must specify the mesh and geometry entities in the adaptive domain that need to be preserved after each remeshing operation. Depending on your model, you might need to preserve:

- One or more node sets, each containing a group of discrete nodes; this approach is necessary for prescribing boundary conditions, or concentrated loads, or constraints, and/or requesting output.

  For example, in *Figure 5*, boundary conditions are applied to the discrete corner nodes. The geometric locations of the nodes in the node set are preserved during remeshing, although the individual node numbers (labels) in the updated mesh might be different after each remeshing operation.
- One or more element sets; this approach is necessary to identify the regions (volumes) for different section definitions, as illustrated in *Figure 5*; to apply volume-based loads, such as gravity; to apply node-based constraints (multi-point constraints, displacement boundary conditions); and to request node-based output on a cloud of nodes over a certain volume.

- One or more surfaces; this approach is necessary for the application of surface-based constraints and distributed loads, and it is necessary to associate mesh entities (node sets) to mesh geometry entities (edges/surfaces) in the adaptive subdomain, as illustrated in *Figure 4*.

You can retain multiple node sets, element sets, and surfaces in the adaptive subdomain simultaneously.

Input File Usage: Use the following option to identify the node sets (containing a group of discrete nodes) to be retained:

**ADAPTIVE REMESH RETAIN*, TYPE=POINT
List of node set labels in the adaptive subdomain that need to be reconstructed after remeshing

Use the following option to identify the element sets to be retained:

**ADAPTIVE REMESH RETAIN*, TYPE=VOLUME
List of element set labels in the adaptive subdomain that need to be reconstructed after remeshing

Use the following option to identify the surfaces to be retained:

**ADAPTIVE REMESH RETAIN*, TYPE=SURFACE
List of user-defined surface labels in the adaptive subdomain that need to be reconstructed after remeshing.

Establishing a Relationship between a Mesh Entity with a Mesh Geometry Entity in the Adaptive Subdomain

In the discussion that follows, the term mesh entity refers to a node set, while the term mesh geometry entity refers to a surface or an edge.

While you can retain the discrete geometric points associated with a node set directly, you should minimize the use of such an approach to provide maximum flexibility to the remeshing algorithm.

As an extreme example, if you were to retain all the node sets (containing all nodes in the model) directly, there would be no changes to the mesh topology during adaptive remeshing. Therefore, you should retain node sets containing a few discrete nodes only to apply boundary conditions, loads, constraints, and/or to request output. If you want to apply boundary conditions or constraints to an edge or a surface, it is recommended that you retain the edge or the surface directly, rather than retaining the corresponding node sets. You can define a surface-based node set to associate mesh entities (node sets) with the retained mesh geometry entities (surfaces or edges). After remeshing, Abaqus/Standard transfers loads, constraints, boundary conditions, and other prescribed conditions automatically from the old mesh to the new mesh using a geometry attachment approach.

Input File Usage: Use the following option to define a surface-based node set:

**NSET*, NSET=*nset_name*, SURFACE=*surf_name*

You can specify the name of a previously defined surface (*About Surfaces*), in which case the nodes on the surface are assigned to the specified node set. You can use this method to define sorted node sets only.

Loads and Boundary Conditions in the Adaptive Subdomain

Within the adaptive subdomain, you cannot apply loads, constraints, nodal transformations, boundary conditions, other prescribed conditions, or request output using individual node or element labels directly because

Abaqus/Standard modifies the node and element labels after each remeshing operation. Instead, you must specify such attributes on mesh or geometric entities such as node sets, element sets, and surfaces. The mesh or geometric entities must be preserved during adaptive remeshing, although the set members are updated after each remeshing operation.

There are no restrictions on applying loads, boundary conditions, or other prescribed conditions in the nonadaptive subdomain.

Applying Boundary Conditions on an Edge or Surface Using the Geometry Attachment Approach

The geometry attachment approach is typically used to apply loads, constraints, nodal transformation, boundary conditions, and other prescribed conditions. It is also used to request output at all the nodes on an edge or a surface.

In this approach, the nodes on an edge or a surface are considered soft entities. The remeshing algorithm preserves the shape of the underlying mesh geometric entity (that is, an edge or a surface), but the mesh density is allowed to change. An example of this involves the application of boundary constraints to an entire surface or edge in the adaptive domain of the model, as illustrated in *Figure 6*.

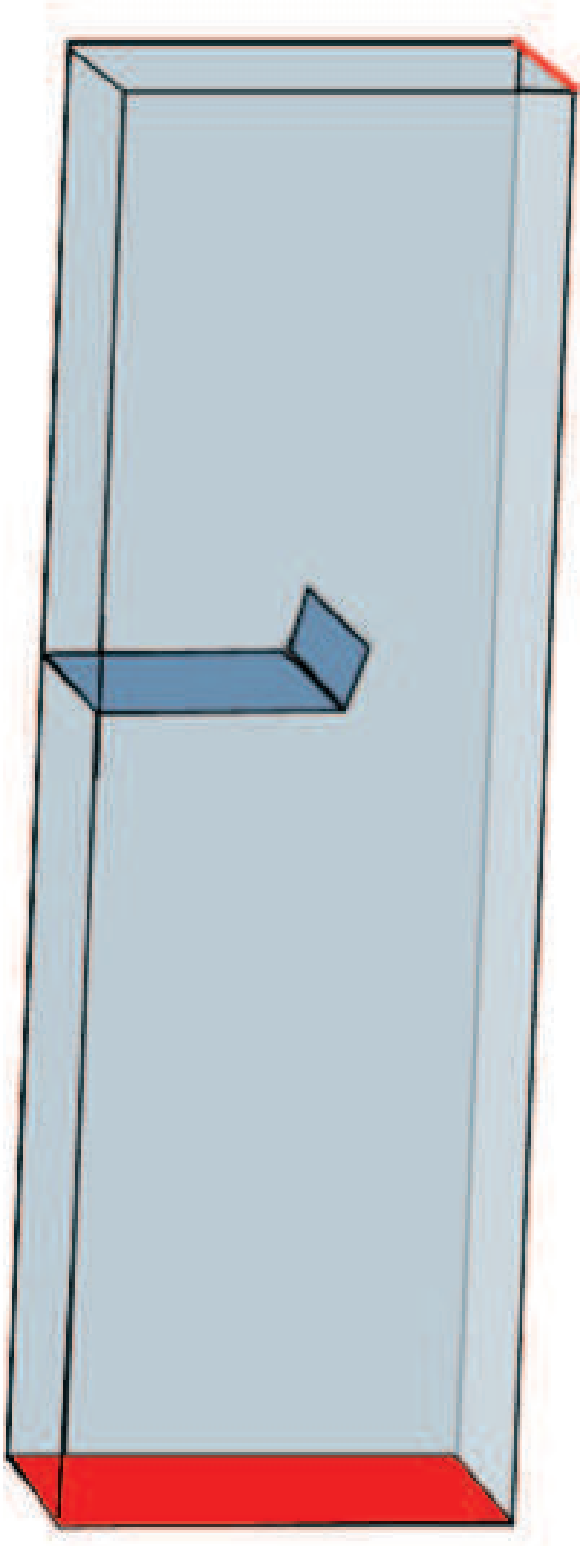


Figure 6: Boundary conditions applied to the bottom surface and the upper right edge of the model.

In this case, the node sets are associated with either an entire edge or an entire surface. The edge and the surface can be defined using either a node-based surface definition or an element-based surface definition. The two mesh geometric entities shown in *Figure 6* (the upper right edge and the bottom surface) are preserved and carried over to the new mesh after remeshing. In this example, Abaqus/Standard updates the members of the node sets on both the edge and the surface based on the new mesh after each remeshing operation.

With the geometry attachment approach for applying boundary conditions on an edge or a surface, you must first identify the edge or the surface that must be retained after remeshing (see *Specifying the Retained Sets and Surfaces in the Adaptive Subdomain of the Model*). Then, you must define the node sets corresponding to these geometric entities using the surface-based node set definition method discussed earlier. This node set contains all the nodes forming an edge already defined with the node-based surface definition, and/or it contains all the nodes forming the surface already defined with the element-based surface definition. You then apply the boundary conditions to such node sets in the usual manner. When you specify a node-based surface definition, you must use an unsorted node set. Abaqus/Standard lists the members of an unsorted node set in order from one end of the edge to the other end of the edge.

Input File Usage: Use the following options to apply boundary conditions to the upper right edge as illustrated in *Figure 6*:

```
*NSET, NSET=top_edgeN, UNSORTED
List of nodes in order from one end of the edge to the other end of the edge
*SURFACE,  TYPE=NODE, NAME=top_edgeS
top_edgeN
*ADAPTIVE REMESH RETAIN, TYPE=SURFACE
top_edgeS
*NSET, NSET=nset_top_edge, SURFACE=top_edgeS
No data lines needed
*BOUNDARY
nset_top_edge,1,3
```

Use the following options to apply boundary conditions to the bottom surface as illustrated in *Figure 6*:

```
*SURFACE, TYPE=ELEMENT, NAME=bot_surfaceS
List of element facets and face identifiers at the bottom surface
*ADAPTIVE REMESH RETAIN, TYPE=SURFACE
bot_surfaceS
*NSET, NSET=nset_bot_surface, SURFACE=bot_surfaceS
No data lines needed
*BOUNDARY
nset_bot_surface,1,3
```

Applying Boundary Conditions on Discrete Nodes Using the Defined Set Approach

The defined set approach is typically used to define loads, constraints, and boundary conditions on a group of a few individual discrete nodes with fixed geometric locations in the adaptive subdomain of the model. They are considered as hard entities. These nodes must be retained to ensure that they are preserved after remeshing.

An example of the application of boundary conditions at a few points in the model is illustrated in *Figure 7*.

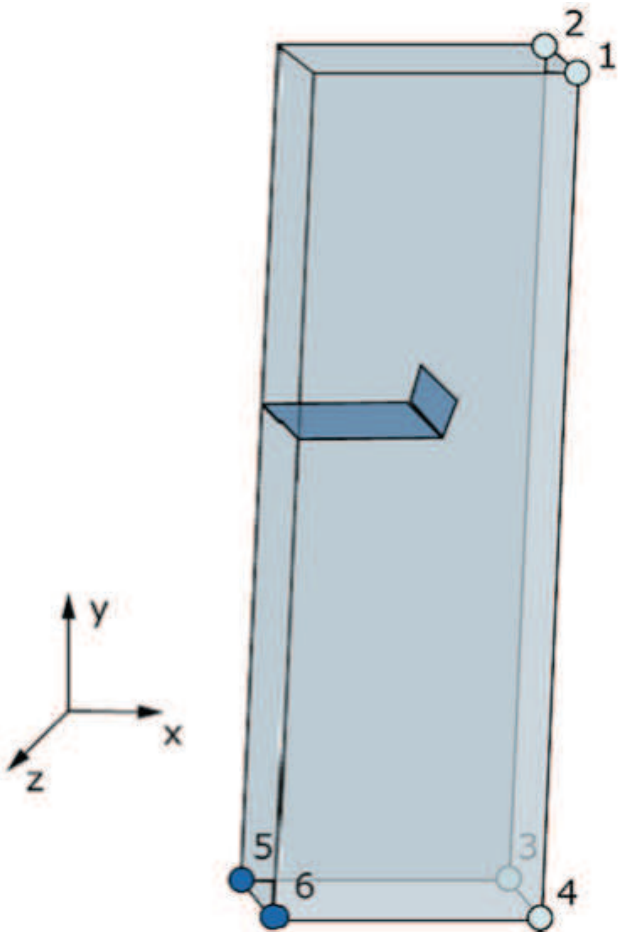


Figure 7: Boundary conditions applied to the vertices of a model.

In this case, you must retain one or more node sets containing these discrete nodes. After remeshing, Abaqus/Standard preserves the geometric location of each individual node specified in such node sets, although the node label (number) of the individual nodes in the node sets might change. The discrete nodes can be located at any geometric location, not only at the corners of the adaptive subdomain.

You must specify the node set(s) that need to be retained after remeshing (see *Specifying the Retained Sets and Surfaces in the Adaptive Subdomain of the Model*). You then apply the boundary conditions to the retained node sets in the usual manner.

Input File Usage: Use the following options to apply boundary conditions to the discrete nodes shown in *Figure 7*:

```
*NSET, NSET=nset_NodeX
1,2,3,4
*NSET, NSET=nset_NodeY
3,4,5,6
*ADAPTIVE REMESH RETAIN, TYPE=POINT
nset_NodeX, nset_NodeY
*BOUNDARY
nset_NodeX,1,1
nset_NodeY,2,2
```

Applying Initial Conditions to all Nodes in the Adaptive Subdomain

If your application requires you to specify initial conditions (for example, on temperature) on all the nodes in the adaptive subdomain, you must first define a node set containing all these nodes. Abaqus/Standard retains such a node set automatically after remeshing. After each remeshing operation, Abaqus/Standard reconstructs the node set that contains all the nodes in the entire adaptive subdomain and also retains the name of the node set.

Input File Usage: Use the following options to apply initial conditions on temperature on all the nodes in the adaptive subdomain of a model (in this case, you do not need to retain this node set explicitly; Abaqus/Standard retains it by default)

**NSET*, NSET=*Nset_all_Adaptive_Subdomain*
List of nodes in the entire adaptive subdomain of the model
**INITIAL CONDITIONS*, TYPE=TEMPERATURE
Nset_all_Adaptive_Subdomain, Temperature Value

Applying Distributed Loads to Element-Based Surfaces in the Adaptive Subdomain

You cannot apply loads, constraints, and other prescribed conditions to any element in the adaptive subdomain using the element label (number) directly. Instead, you must use an element-based surface definition to apply distributed loads or constraints to a group of element facets. *Figure 8* shows an example of an application where surface-based distributed loading needs to be applied to the two shaded surfaces.

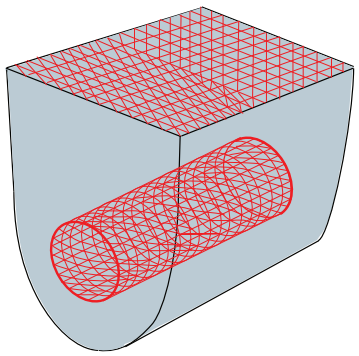


Figure 8: Distributed loads applied to the top surface and inner surface of the model.

Clearly these surfaces must be retained following each adaptive remeshing operation. First, you must specify these surfaces as retained (see *Specifying the Retained Sets and Surfaces in the Adaptive Subdomain of the Model*); then, you apply the distributed loads to these surfaces in the usual manner. While Abaqus/Standard retains the specified surface, the individual element facets that make up this surface are updated with every remeshing operation. Abaqus/Standard also retains the surface name. This algorithm is subject to the restriction that any element facet cannot be a part of more than one retained surface definition in the adaptive subdomain.

When you specify an element-based surface that needs to be retained, Abaqus/Standard does not retain the element sets used to define the original surface. Instead, after remeshing, the data lines of the surface definition option are converted into updated individual element labels and face identifiers that replace the old set that existed prior to remeshing.

Input File Usage: Use the following options to apply distributed loads to the top and the inner surfaces in *Figure 8*:

**SURFACE*, TYPE=ELEMENT, NAME=*TopSurf_name*
Element set name or element label, face identifiers
**SURFACE*, TYPE=ELEMENT, NAME=*InnerSurf_name*
Element set name or element label, face identifiers
**ADAPTIVE REMESH RETAIN*, TYPE=SURFACE
TopSurf_name, *InnerSurf_name*
**DSLOAD*
TopSurf_name
InnerSurf_name

Defining Multiple Solid Section Properties in the Adaptive Subdomain

By default, after each remeshing operation, Abaqus/Standard retains certain element sets in the adaptive subdomain. An example of an element set that is retained by default is one that contains all the elements in the adaptive subdomain. Such an element set is typically used to define the element section or element-based volume loads (such as body forces) for the entire adaptive subdomain.

If the adaptive subdomain requires more than one solid section property definition in the model (see *Figure 5*), the element set corresponding to each section definition must be retained after each operation (see *Specifying the Retained Sets and Surfaces in the Adaptive Subdomain of the Model*). Other situations that require retaining multiple element sets in the adaptive subdomain include the application of different element-based body forces to different parts of the adaptive subdomain, the application of node-based constraints (MPCs or displacement boundary conditions), or the need to request node-based output on a cloud of nodes over a certain material volume.

If it is necessary to retain more than one element set, the following apply:

- Abaqus/Standard preserves the volume occupied by each element set in the model after remeshing.
- You cannot specify any given element in more than one retained element set.

Retaining multiple element sets is equivalent to partitioning the adaptive subdomain into several parts. Ideally, you should minimize retaining element sets as much as possible for maximum flexibility with remeshing.

Input File Usage: Use the following options to assign all the elements in the adaptive subdomain to a single element set (in this case, you do not need to retain this element set explicitly; Abaqus/Standard retains it by default):

**ELSET*, ELSET=*Elset_all_Adaptive_Subdomain*
List of elements in the entire adaptive subdomain of the model
**SOLID SECTION*, ELSET=*Elset_all_Adaptive_Subdomain*

Use the following options to preserve multiple regions (material volumes) corresponding to different section definitions, or to apply volume-based loads (such as gravity) in a portion of the adaptive subdomain, as shown in *Figure 5*:

**ELSET*, ELSET=*Elset_name1*
List of elements in part one of the adaptive subdomain
**ELSET*, ELSET=*Elset_name2*
List of elements in part two of the adaptive subdomain
**ELSET*, ELSET=*Elset_name3*
List of elements in part three of the adaptive subdomain
**ADAPTIVE REMESH RETAIN*, TYPE=VOLUME

Elset_name1, *Elset_name2*, *Elset_name3*
**SOLID SECTION*, ELSET=*Elset_name1*
**SOLID SECTION*, ELSET=*Elset_name2*
**SOLID SECTION*, ELSET=*Elset_name3*

Use the following options to request node-based output on a cloud of nodes over a certain material volume in the adaptive subdomain:

**ELSET*, ELSET=*Elset_name*
List of elements in a certain material volume within the adaptive subdomain
**ADAPTIVE REMESH RETAIN*, TYPE=VOLUME
Elset_name
**NSET*, NSET=*Nset_name*, ELSET=*Elset_name*
No data lines needed
**NODE OUTPUT*, NSET=*Nset_name*

Fatigue Crack Growth Criterion

Simulation of fatigue crack growth under subcritical cyclic loading requires the specification of a fatigue crack growth criterion. For fatigue crack growth with adaptive remeshing, you can use the criterion only with a general linear elastic fatigue crack growth analysis (see *Linear Elastic Fatigue Crack Growth Analysis*). You cannot use fatigue crack growth with adaptive remeshing for low-cycle fatigue applications using the direct cyclic approach. A fatigue crack growth step can be the only step in the analysis, or it can follow a general static step that is used for preloading the structure prior to the fatigue analysis. You cannot use multiple fatigue crack growth steps in a single simulation.

The onset and growth of a fatigue crack are characterized by the Paris law, which relates the initiation as well as the rate of crack growth to the fracture energy release rate or the stress intensity factor. Different forms of the basic Paris law are available in Abaqus/Standard. For more information, see *Linear Elastic Fatigue Crack Growth Behaviors*.

Input File Usage: The option to specify a fatigue crack growth law is part of the material definition and must appear there along with other relevant material options (properties):

**MATERIAL*
**FRACTURE CRITERION*, TYPE=FATIGUE

Contour Integral Evaluation

Abaqus/Standard calculates the fracture energy release rates or the stress intensity factors at the crack-tip nodes based on the contour integral method (see *Contour Integral Evaluation*). Only second-order tetrahedral elements can be used to mesh the adaptive domain.

The mesh is unstructured (as is typical for tetrahedral meshes); therefore, the mesh around the crack front must be refined with several layers or rings of elements to obtain accurate results from the contour integral evaluations.

If the crack front intersects the external surface of a three-dimensional solid, the virtual crack extension direction, q, must lie in the plane of the free surface for accurate contour integral evaluation. To achieve this if the free surface is planar and the normal to the free surface remains unchanged during the fatigue crack growth, you can specify the surface normals directly by using the nodal coordinates (the fourth–sixth coordinates) at the crack-tip nodes lying on the free surface. If the free surface is nonplanar and the normal on the free surface intersected with the crack front is expected to change, you can request that Abaqus/Standard calculates the normal for these nodes to correct any inadequate virtual crack extension direction. For large models, requesting that Abaqus/Standard calculate the surface normals on the free surfaces can increase the computation time.

Input File Usage: Use both of the parameters listed below if the normals on the free surfaces are specified by the user:

**CONTOUR INTEGRAL*, TYPE=K FACTORS, RING RADIUS

Use all three parameters listed below to specify that the normals to the free surfaces should be calculated by Abaqus/Standard:

**CONTOUR INTEGRAL*, TYPE=K FACTORS, RING RADIUS, SURFACE NORMAL=FREE

Controlling the Incremental Crack Growth

Fatigue crack growth is characterized by the Paris law, which relates the crack growth rate to the fracture energy release rate or the stress intensity factor (see *Fatigue Crack Growth Using the Paris Law*):

$$\frac{da}{dN} = c\Delta K^m.$$

In addition, you can utilize user subroutine *UMIXMODEFATIGUE* to implement a user-defined fatigue crack growth law. The Paris regime is bounded by the stress intensity factor fatigue threshold, K_{thresh}, below which there is no consideration of fatigue crack initiation or growth, and the stress intensity factor fatigue upper limit, K_{pl}, above which the fatigue crack grows at an accelerated rate. The process of fatigue crack growth is based on a damage extrapolation technique, which relies on propagating a crack tip node by at least a factor of a typical element characteristic length ahead of the current crack front (once the onset criterion is satisfied) with a corresponding increase in the number of cycles. Specifically, if the criterion for the onset of the crack growth is satisfied at any crack tip at the end of a completed cycle, N, Abaqus/Standard extends the crack length Δa_N, by propagating it by at least a factor of a typical element characteristic length ahead of the current crack front. This local (at each crack front node) increase in the crack length corresponds to an increase in the cycle count from N to $N + \Delta N$, where ΔN is computed based upon the Paris Law. If $K_{\max} > K_{pl}$, the elements ahead of the crack tips are fractured by increasing the cycle number count, dN, by one only.

Two different approaches are available to control the propagation of the crack: a cycle increment-based approach and a crack length increment-based approach.

Input File Usage: Use the following option to specify a fatigue crack growth analysis with a cycle increment-based or a crack length increment-based approach.

**FATIGUE*
initial time increment, single loading cycle time, minimum time increment, maximum time increment
$\Delta N_{\min}$, $\Delta N_{\max}$, N_{total}, α

When you specify a nonzero value of α, Abaqus/Standard uses the crack length increment–based approach.

Cycle Increment-Based Approach

You specify the minimum and maximum number of cycles ($\Delta N_{\min}$ and $\Delta N_{\max}$) over which the crack is extended.

Abaqus/Standard then initially sets ΔN equal to $\Delta N_{\max}/2$ to calculate the crack extension length Δa_N for all the nodes along the crack front.

If none of the calculated Δa_N for the nodes along the current crack front is equal to or greater than one typical element characteristic length, Abaqus/Standard adjusts the cycle increment ΔN to ensure that the crack propagates at least one typical element characteristic length for at least one of the nodes along the crack front.

On the other hand, if any calculated Δa_N for the nodes along the current crack front is greater than two times the typical element characteristic length, Abaqus/Standard adjusts the cycle increment ΔN to ensure none of the nodes along the crack front propagates more than two times the element characteristic length.

Crack Length Increment-Based Approach

To use the crack length increment-based approach, you specify a nonzero value for α, the ratio of the crack extension length, Δa_N to the typical element characteristic length. Abaqus/Standard extends the crack length by Δa_N for all the crack-tip nodes and calculates the corresponding cycle increment ΔN_i for each node i along the crack front. The overall cycle increment for this increment of crack growth, ΔN, is then set equal to the average of ΔN_i. When you use the crack length increment-based approach, the crack shape always grows in a self-similar manner although the size of the crack can increase.

Performing Adaptive Remeshing after Each Crack Extension

At the beginning of the workflow, Abaqus/Standard automatically extracts the following data and creates two auxiliary files:

- The node, element, node and element set, and surface definitions within the adaptive subdomain
- The crack front information (that is, the crack front nodes and the direction cosines defining the crack propagation direction at each node along the crack front) within the adaptive subdomain

Next, Abaqus/Standard modifies the original input file for the model by replacing the node, element, set, surface, and crack front definitions in the adaptive subdomain with the corresponding data contained in the two auxiliary files. This is accomplished by 'including' (*Including Model or History Data from an External File*) the two auxiliary files in the modified input file. As the crack grows and the mesh updates adaptively, the contents of the two auxiliary files also update for the next phase of crack growth.

The updated crack front location is obtained using the procedure described in the earlier sections. Abaqus/Standard uses a tetrahedral element-based adaptive remeshing technology to morph the old mesh to a new mesh, such that it conforms to the updated crack front. As discussed above, Abaqus/Standard writes a new input file, along with two external include files (that are referenced from the new input file), with the updated mesh. In the nonadaptive subdomain, all options and their associated data lines remain unchanged in the updated input file following each adaptive remeshing operation. To avoid the risk of potentially conflicting element and node labels in the model after each remeshing operation, the largest element label and the largest node label in the nonadaptive subdomain are used as offsets when generating new node and new element labels, respectively, in the adaptive subdomain. Care must be taken that any large offset values will not result in a generated node or element label in the adaptive subdomain exceeding the maximum value allowed, which is 999,999,999.

Execution and File Management

Fatigue crack growth with adaptive remeshing is based upon a series of automated successive Abaqus/Standard fatigue analyses with progressively longer crack lengths. You must launch the workflow execution using the following command:

```
abaqus -j jobname -input filename -evol adap -cpus  ncpus -thread  nthreads
```

You must specify the job name and the file name separately. Abaqus/Standard generates a number of files over the course of the workflow. Some of these files contain analysis results and intermediate input files; Abaqus/Standard retains these files for use by other analysis options to maintain the flexibility of this workflow.

For example, from an intermediate point in the workflow (which corresponds to the cracked structure after it has been subjected to many fatigue loading cycles) you can perform:

- a stand-alone analysis to evaluate the linear or nonlinear static or eigenfrequency responses of the cracked structure;
- a restart-based fatigue analysis using a different cyclic loading amplitude compared to the one specified in the original analysis; or
- additional postprocessing.

If you perform either a stand-alone analysis or a restart-based fatigue analysis from an intermediate point in the workflow, you must augment the input files retained by Abaqus with either additional steps or a separate restart analysis input file.

This section describes the files that are created and retained by Abaqus/Standard during this workflow.

At the end of each fatigue crack growth increment corresponding to a fatigue cycle number, Abaqus/Standard automatically creates a new input file (`.inp`) and two external (`.inc`) files with updated crack geometry. Abaqus/Standard names these files by appending `_cyclenumber` to the original file names and uses the new files to continue the workflow. Specifically, the new files are the input for a new Abaqus/Standard fatigue crack growth analysis to compute the next increment of crack growth based on the updated crack geometry.

After each fatigue crack growth increment, Abaqus/Standard retains each of the following:

- the newly generated input file (`.inp`),
- the two external files (`.inc`)—see *Including Model or History Data from an External File*, and
- the output data files (`.odb` and `.sim`).

In addition, Abaqus/Standard retains a single file of each of the following types for the entire analysis:

- printed job output file (`.dat`),
- log file (`.log`),
- message file (`.msg`),
- usage file (`.use`), and
- status file (`.sta`).

To do so, it progressively appends the files generated at the end of the current fatigue crack growth cycle increment to those obtained at the end of the previous fatigue crack growth cycle increments.

Other files exist only while Abaqus/Standard is executing; Abaqus/Standard deletes them once the workflows completes.

Visualization

A special Abaqus/Viewer plug-in is available to:

- create a series of frames (`.png` files) to show the deformed shapes of the structures following each increment of crack growth;
- facilitate the creation of path plots for the series of crack growth analyses; and
- create animations of crack growth in MP4 or GIF format directly.

For more information about the plug-in, see Dassault Systèmes Knowledge Base item QA00000428074 (*https://support.3ds.com/knowledge-base/?q=docid:QA00000428074*).

Procedures

The adaptive remeshing capability is available only for linear elastic fatigue crack growth analysis (see *Linear Elastic Fatigue Crack Growth Analysis*).

You cannot use multiple fatigue steps in a single simulation.

Initial Conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see *Initial Conditions*) with some restrictions (see *Loads and Boundary Conditions in the Adaptive Subdomain*).

Boundary Conditions

Boundary conditions can be applied to any of the displacement degrees of freedom (see *Boundary Conditions*) with some restrictions (see *Loads and Boundary Conditions in the Adaptive Subdomain*).

Loads

Concentrated nodal forces can be applied to the displacement degrees of freedom (see *Concentrated Loads*) with some restrictions (see *Loads and Boundary Conditions in the Adaptive Subdomain*).

Distributed pressure loads or body forces associated with tetrahedral continuum elements can be applied (see *Distributed Loads*) with some restrictions (see *Loads and Boundary Conditions in the Adaptive Subdomain*).

Predefined Fields

Predefined temperatures and field variables can be specified in a model, as described in *Predefined Fields*, with some restrictions (see *Loads and Boundary Conditions in the Adaptive Subdomain*). The specified values will affect temperature- and field-variable-dependent material properties, if any.

Material Options

The adaptive remeshing capability can be used only with the linear elastic material model (see *Elastic Behavior*).

Elements

The adaptive remeshing capability can be used only with second-order tetrahedral stress/displacement continuum elements in the adaptive subdomain.

You can use any of the stress/displacement elements available in the Abaqus/Standard element library in the nonadaptive subdomain (see *Choosing the Appropriate Element for an Analysis Type*).

Limitations

- You can only use a mesh with second-order tetrahedral elements in the adaptive subdomain.
- You cannot use fatigue crack growth with adaptive remeshing in a model defined in terms of an assembly of part instances.
- No state mapping is performed (or necessary) between Abaqus/Standard analyses to compute successive increments in crack growth.
- You can include only a single crack in the adaptive subdomain of the model.
- You cannot use multiple fatigue steps in a single simulation.
- Mesh morphing fails (hence, the workflow terminates) when an embedded crack tip has propagated to the free/external surface of the model.

Input File Template

The following is an example of a typical input file for modeling fatigue crack propagation with adaptive remeshing for the model illustrated in *Figure 3*:

```
*HEADING
*NODE
*NSET, NSET=NSET_INNER
*NSET, NSET=NSET_OUTER
*ELEMENT, TYPE=C3D10
**Define the element set in the adaptive domain
*ELSET, ELSET=ELSET_INNER
**Define the element set in the nonadaptive domain
*ELSET, ELSET=ELSET_OUTER
**Specify an outer surface of the adaptive domain
*ELSET, ELSET=elset_name1
*ELSET, ELSET=elset_name2
*SURFACE, TYPE=ELEMENT, NAME=OUTER_SURF
elset_name1
elset_name2
**Specify an inner surface of the nonadaptive domain
*ELSET, ELSET=elset_name3
*ELSET, ELSET=elset_name4
*SURFACE, TYPE=ELEMENT, NAME=INNER_SURF
elset_name3
elset_name4
**Specify additional surface in the adaptive domain
**subjected to internal pressure
*ELSET, ELSET=elset_name5
*ELSET, ELSET=elset_name6
*SURAFCE, TYPE=ELEMENT, NAME=PRESSURE_SURF
elset_name5
elset_name6
**Identify the adaptive domain
*ADAPTIVE REMESH, ELSET=ELSET_INNER
**Identify the surfaces in the adaptive subdomain
**that need to be reconstructed after remeshing
*ADAPTIVE REMESH RETAIN, TYPE=SURFACE
OUTER_SURF, PRESSURE_SURF
**Specify node sets with a few discrete nodes in the nonadaptive domain
*NSET, NSET=nset_NodeX
*NSET, NSET=nset_NodeY
*SOLID SECTION, MATERIAL=STEEL, ELSET=ELSET_INNER
*SOLID SECTION, MATERIAL=IRON, ELSET=ELSET_OUTER
*MATERIAL, NAME=STEEL
**Specify the crack growth law
*FRACTURE CRITERION, TYPE=FATIGUE
Data lines to specify the fatigue crack propagation criterion
*MATERIAL, NAME=IRON
**Impose tie constraints between the outer surface of the adaptive domain
**and the inner surface of the nonadaptive domain
*TIE
OUTER_SURF, INNER_SURF
*STEP
**Define a fatigue crack growth step
*FATIGUE
```

```
…
*BOUNDARY
nset_nodeX
nset_nodeY
**Apply pressure to an internal surface in the adaptive domain
*DSLOAD
PRESSURE_SURF, P
**Perform contour integral calculations
*CONTOUR INTEGRAL, TYPE=K FACTORS, RINGRADIUS, SURFACE NORMAL=FREE
Nodeid_1, nodeid_1, qx1, qy1, qz1
Nodeid_2, nodeid_2, qx2, qy2, qz2
Nodeid_3, nodeid_3, qx3, qy3, qz3
……
*OUTPUT, FIELD
*NODE OUTPUT
K, U
*END STEP
```

Optimization Techniques

In this section:

- *About Structural Optimization*
- *Optimization Models*

About Structural Optimization

Overview

Structural optimization using Abaqus is an iterative process that helps you refine your designs. The result of a well-designed structural optimization is a component that is lightweight, rigid, and durable.

Abaqus provides the following approaches to structural optimization: topology optimization, shape optimization, sizing optimization, and bead optimization. Topology optimization starts with an initial model and determines an optimum design by modifying the properties of the material in selected elements, effectively removing elements from the analysis. Shape and sizing optimization further refine the model. Shape optimization modifies the surface of the component by moving the surface nodes to reduce local stress concentrations. Sizing optimization modifies the sheet thickness of sheet metal components; typically, to increase the stiffness or reduce vibration. Bead optimization creates stiffening beads in a shell model. Topology, shape, sizing, and bead optimization are governed by a set of objectives and constraints.

Optimization is a tool for shortening the development process by adding value to a designer's experience and intuition with an automated procedure. To optimize your model, you need to know what to optimize. It is not sufficient to say that you want to minimize stresses or maximize eigenvalues, your statements must be more specific. For example, you might want to minimize the maximal nodal stresses experienced during two load cases. Similarly, you might want to maximize the sum of the first five eigenvalues. The goal of an optimization is called the objective function. In addition, you can enforce certain values during the optimization. For example, you can specify that the displacement of a given node must not exceed a certain value. An enforced value is called a constraint.

You use Abaqus/CAE to create the model to be optimized and to define, configure, and execute the structural optimization. For more information, see *The Optimization module*.

Terminology

Structural optimization introduces its own terminology. The following terms are used throughout the Abaqus documentation and the Abaqus/CAE user interface:

Design area

The design area is the region of your model that the structural optimization modifies. The design area can be the whole model, or it can be a subset of the model containing only selected regions. Given the prescribed conditions (such as boundary conditions, loads, and manufacturing constraints),

- a topology optimization process removes and adds material from elements in the design area while it attempts to reach an optimal design,
- a shape optimization modifies the surface of the design area by moving surface nodes,
- a sizing optimization modifies the thickness of the design area by changing the thickness of shell elements, and
- a bead optimization moves nodes of shell elements in the design area in the direction of the shell normal.

Design variables

For an optimization problem, the design variables represent the parameters to be changed during the optimization.

For a topology optimization, the densities of the elements in the design area are the design variables. The Optimization module changes the density during each iteration of the optimization and couples the stiffness of each element with the density. In effect, the optimization removes elements from your model by giving them a mass and stiffness that is small enough to ensure they no longer participate in the overall response of the structure. The model with the revised material properties is then analyzed by Abaqus.

For a shape optimization, the displacements of the surface nodes in the design area are the design variables. During the optimization, the Optimization module either moves a node outward (growth) or inward (shrinkage) or leaves the position unchanged (neutral). Restrictions influence the amount a surface node can move and the direction in which it can move. The optimization directly modifies only the position of the corner nodes of elements; the Optimization module interpolates the displacement of midside nodes from the movement of the corner nodes.

For a sizing optimization, the thicknesses of the shell elements in the design area are the design variables. The Optimization module can adjust the thickness of individual shell elements, or you can require clustering—the simultaneous modification of shell thicknesses in specific areas.

For a bead optimization, the displacements of the nodes of the shell elements that form the stiffening beads in the design area are the design variables. Restrictions limit the amount a node can move and the direction in which it can move.

Design cycle

Optimization is an iterative design process that updates the design variables, executes an Abaqus analysis of the modified model, and reviews the results to determine if an optimized solution has been reached. Each optimization iteration is called a design cycle.

Optimization task

An optimization task contains the definition of your optimization, such as the design responses, objectives, constraints, and geometric restrictions. To run an optimization, you execute an optimization process. An optimization process refers to an optimization task.

Design responses

The inputs to the optimization are called the design responses. Design responses can be read directly from the Abaqus output database (`.odb`) file; for example, stiffness, stress, eigenfrequencies, and displacements. Alternatively, the Optimization module can read data from the output database file and calculate the design responses from your model; for example, its weight, center of mass, or relative displacements.

A design response is associated with a region of your model; however, it consists of a single scalar value, such as the maximum stress within a region or the total volume of the model. In addition, a design response can be associated with a particular step or load case.

Objective functions

Objective functions define the objective of the optimization. An objective function is a single scalar value extracted from a design response, such as the maximum displacement or the maximum stress. An objective function can be formulated from multiple design responses. If you specify that the objective functions minimize or maximize the design responses, the Optimization module calculates the objective function by adding each of the values determined from the design responses. In addition, if you have multiple objective functions, you can use a weighting factor to scale their influence on the optimization.

Constraints

Constraints are also a single scalar value extracted from a design response; however, a constraint cannot be derived from a combination of design responses. Constraints restrict the value of a design response; for example, you can specify that the volume must be reduced by 45% or the absolute displacement in a region must not exceed 1 mm. You can also apply manufacturing and geometric constraints that are independent of the optimization; for example, a structure must be able to be cast or stamped or the diameter of a bearing surface cannot be changed.

Stop conditions

A global stop condition defines the maximum number of iterations an optimization can perform. A local stop condition specifies that the optimization should end when a local minimum (or maximum) has been reached.

Structural Optimization with Abaqus/CAE

The following steps are required to incorporate structural optimization into your Abaqus/CAE model:

- You create an Abaqus model that can be optimized. For example, the design area must include only supported elements and materials. See *Creating Abaqus Optimization Models*.
- You create an optimization task. See *Creating and configuring an optimization task*.
- You create design responses. See *Design Responses*.
- You use the design responses to create objective functions and constraints. See *Objectives and Constraints*.
- You create an optimization process and submit it for analysis. See *What is an optimization process?*.

Based on the definition of the optimization task and the optimization process, the Optimization module iteratively:

- prepares the design variables (element densities or surface node positions) and updates the Abaqus finite element model, and
- executes an Abaqus/Standard analysis.

These iterations or design cycles continue until either:

- the maximum number of design cycles is reached, or
- the specified stop conditions are reached.

Figure 1 shows the interaction of Abaqus and the optimization process.

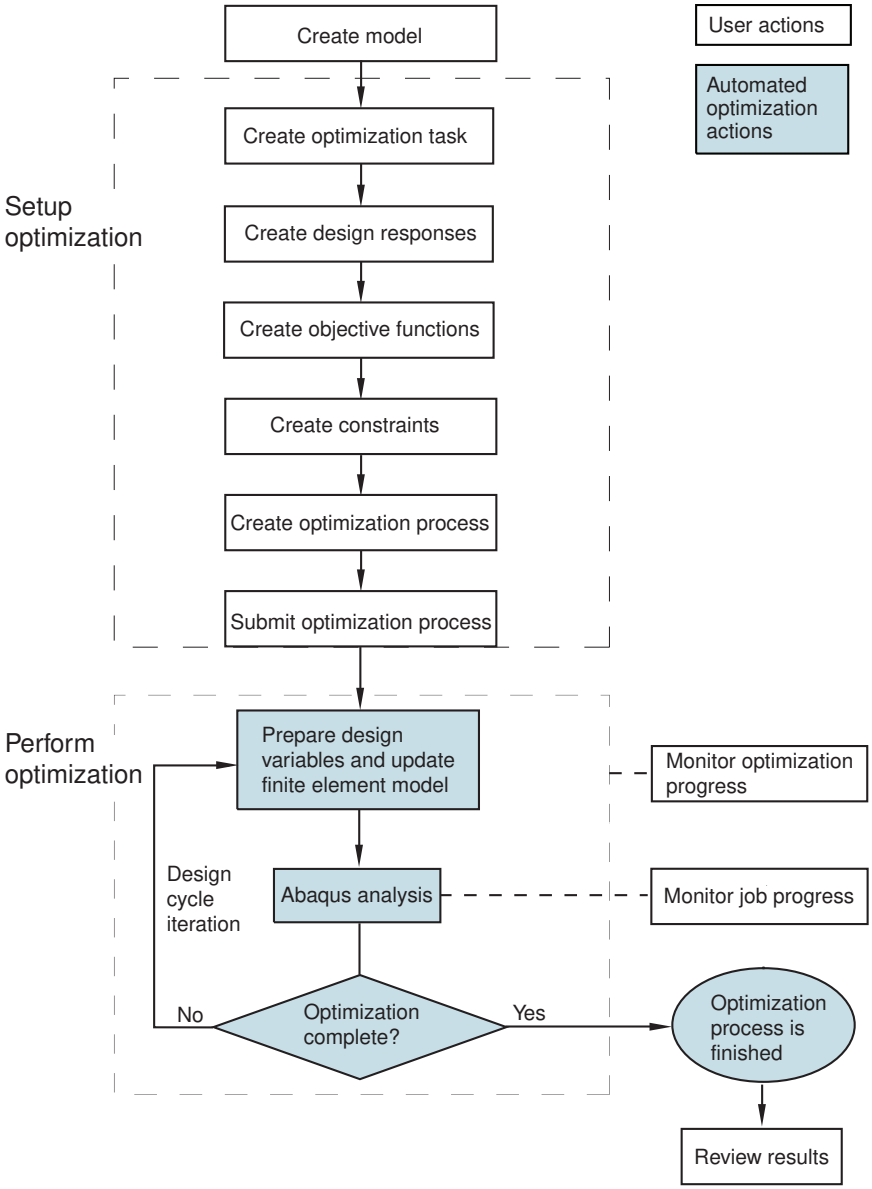


Figure 1: User actions and automated Abaqus/CAE actions in the optimization process.

Topology Optimization

Topology optimization starts with an initial design (the original design area), which also contains any prescribed conditions (such as boundary conditions and loads). The optimization process determines a new material distribution by changing the density and the stiffness of the elements in the initial design while continuing to satisfy the optimization constraints, such as the minimum volume or the maximum displacement of a region. In addition, you can apply a number of manufacturing constraints that ensure the proposed design can be created using standard production processes, such as casting and stamping. You can also freeze selected regions and apply member size, symmetry, and coupling constraints.

Figure 2 show the progression of a topology optimization of an automotive control arm during 17 design cycles. The objective function in the optimization is trying to minimize the maximum strain energy calculated from all the elements in the arm, in effect maximizing the structural stiffness of the arm. The constraint is forcing the

optimization to reduce the volume by 57% from the initial value. During the optimization the density and the stiffness of the elements in the center of the arm are reduced so that the elements are, in effect, “removed” from the analysis. However, the elements are still present, and they could play a role in the analysis if their density and stiffness increase as the optimization continues. A geometry restriction forces the optimization to create a model that could be cast and removed from a mold—the Optimization module cannot create voids and undercuts.

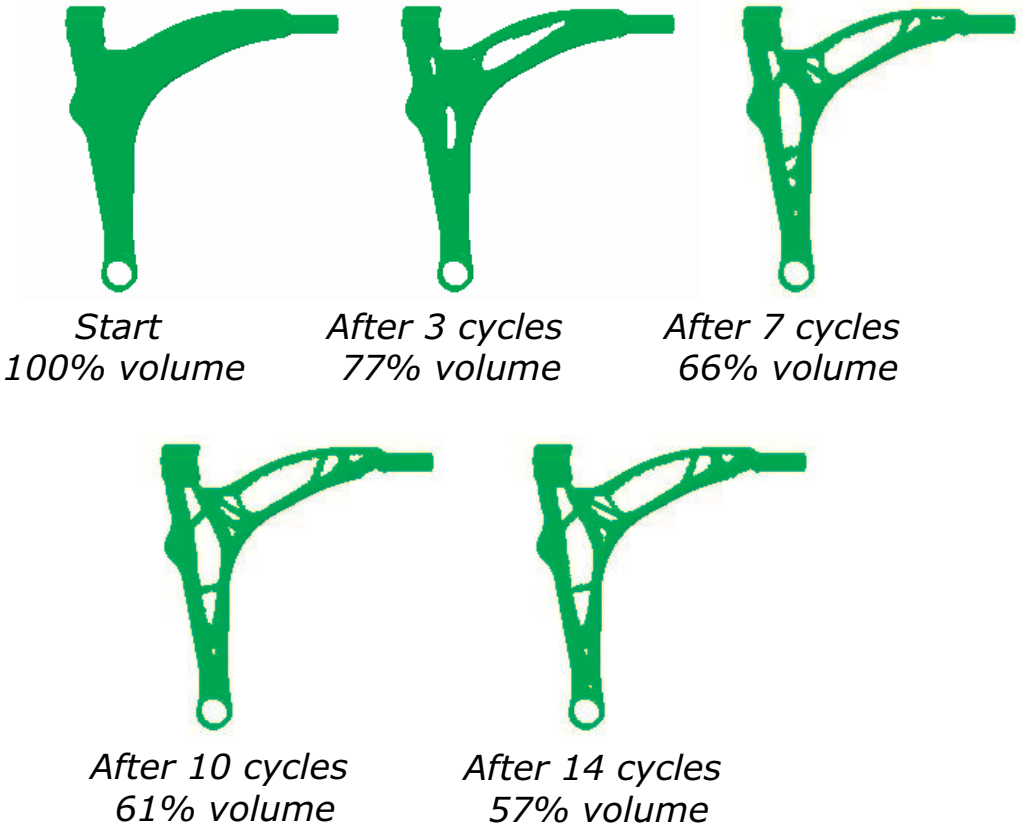


Figure 2: The progression of a topology optimization.

An example of using topology optimization is provided in *Topology optimization of an automotive control arm*. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

General Versus Condition-Based Topology Optimization

Topology optimization supports two algorithms—the general algorithm, which is more flexible and can be applied to most problems, and the condition-based algorithm, which is more efficient but has limited capabilities. By default, the Optimization module uses the general algorithm; however, you can choose which algorithm to use when you create the optimization task. Each algorithm has a different approach for determining the optimized solution.

Algorithms

General topology optimization uses an algorithm that adjusts the density and stiffness of the design variables while trying to satisfy the objective function and the constraints. The general algorithm is partly described in Bendsøe and Sigmund (2003). In contrast, condition-based topology optimization uses a more efficient algorithm that uses the strain energy and the stresses at the nodes as input data and does not need to calculate the local

stiffness of the design variables. The condition-based algorithm was developed at the University of Karlsruhe, Germany and is described in Bakhtiary (1996).

Elements with Intermediate Densities

The general algorithm generates intermediate elements in the final design (their relative density is between zero and one). In contrast, the condition-based optimization algorithm generates elements in the final design that are either void (their relative density is very close to zero) or solid (their relative density is equal to one).

Number of Optimization Design Cycles

The number of design cycles used by the general optimization algorithm is unknown before the optimization starts, but normally the number of design cycles is between 30 and 45. The condition-based optimization algorithm is more efficient and searches for a solution until it reaches the maximum number of optimization design cycles (15 by default).

Analysis Types

The general algorithm supports the responses of linear and nonlinear static and linear eigenfrequency finite element analyses. Both algorithms support geometrical nonlinearities and contact, and many nonlinear materials are also supported.

Furthermore, prescribed displacements are allowed in the Abaqus model for static topology optimization. However, prescribed displacements are not allowed for modal analysis. You can use topology optimization on a structure that uses a composite material; however, the individual laminates of a composite material cannot be modified using topology optimization. For example, you cannot change the orientation of the fibers.

Objective Functions and Constraints

The general topology optimization algorithm can use one objective function and several constraints, where the constraints are all inequality constraints. A variety of design responses can be used to define the objective and the constraints, such as strain energy, displacements and rotations, reaction and internal forces, eigenfrequencies, and material volume and weight. The condition-based topology optimization algorithm is more efficient; however, it is less flexible and supports only strain energy (a measure of stiffness) as the objective function and the material volume as an equality constraint.

Tables in *The Available Design Responses for Each Type of Optimization* list the available design responses for condition-based and general topology optimization and describe which design responses can be used as objectives and/or constraints.

Shape Optimization

Shape optimization uses an algorithm that is similar to the algorithm used by condition-based topology optimization. You use shape optimization at the end of the design process when the general layout of a component is fixed, and only minor changes are allowed by repositioning surface nodes in selected regions. A shape optimization starts with a finite element model that needs minor improvement or with the finite element model generated by a topology optimization.

Typically, the objective of a shape optimization is to minimize stress concentrations using the results of a stress analysis to modify the surface geometry of a component until the required stress level is reached. Shape optimization tries to position the surface nodes of the selected region until the stress across the region is constant (stress homogenization). *Figure 3* shows a region at the base of a connecting rod where the surface nodes have been moved by shape optimization to reduce the effect of a stress concentration.

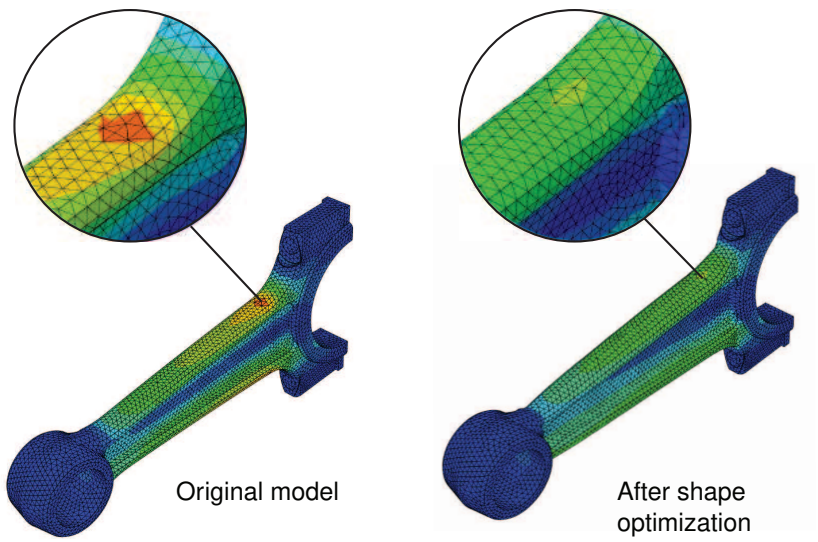


Figure 3: The effect of shape optimization.

A table in *The Available Design Responses for Each Type of Optimization* lists the available design responses for shape optimization and describes which design responses can be used as objectives and/or constraints. In addition, you can apply a number of manufacturing geometric restrictions that ensure the proposed design can continue to be produced using casting or stamping processes. You can also freeze selected regions and apply member size, symmetry, and coupling restrictions.

An example of using shape optimization is provided in *Shape optimization of a connecting rod*. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying Mesh Smoothing to a Shape Optimization

During a shape optimization, the Optimization module modifies the surface of your model. If the Optimization module modifies only the surface nodes without adjusting the inner nodes, the layer of surface elements becomes distorted. Therefore, the results of the Abaqus analysis are no longer reliable, and the quality of the optimization suffers. To maintain the quality of the surface elements, the Optimization module can apply mesh smoothing to selected regions, which adjusts the position of the inner nodes in relation to the movement of the surface nodes. You must have a good quality finite element mesh before you start the shape optimization, especially in areas where you expect the shape to change.

The Optimization module can apply mesh smoothing to the standard continuum elements, such as triangular, quadrilateral, and tetrahedral elements. Other element types are ignored during the mesh smoothing. You can specify the relative quality of the smoothed mesh, and you can specify the range of angles (quadrilateral and triangular elements) or the range of aspect ratios (tetrahedral elements) that define an element that is considered good quality. Elements that are considered poor are given a quality rating. The poorer an element is rated, the greater the consideration it will be given in improving the element quality.

Mesh smoothing can be computationally expensive. The mesh smoothing algorithm is element-based; and computing time increases in regions with many elements with limited degrees of freedom, such as regions with small tetrahedral elements. You should apply mesh smoothing only to regions where you expect the surface nodes to move—regions that will benefit from mesh smoothing. The nodes in the regions to which you apply mesh smoothing must be free to move. For example, you should not apply mesh smoothing to fixed nodes or to frozen regions.

You can apply limits to the result of mesh smoothing by applying minimum and maximum growth restrictions to the selected region. See *Creating a growth restriction* for more information.

Mesh smoothing can be applied to regions that are included in the design region and to regions that are outside the design region. In particular, you can prevent element distortion by applying mesh smoothing to the region of transition between the design region and the rest of your model. However, the design region must be contained within the region to which you apply mesh smoothing.

Free surface nodes are defined as the nodes that lie outside the design area and are not included in a geometric restriction. By default, the Optimization module fixes all degrees of freedom of all of the free surface nodes, and they are not modified during the mesh smoothing operation. Alternatively, you can choose to allow the free surface nodes to move along with a specified number of layers of nodes adjacent to the nodes in the design area. (A "layer" of nodes is created from only corner nodes; midside nodes are not taken into consideration.)

You should allow free surface nodes to move in regions that are adjacent to the design area to create a smooth transition between optimized and non-optimized regions. However, in some cases you will want free surface nodes to remain fixed; for example, on a planar face that does not play a role in your optimized model and must remain planar.

By default, a constrained Laplacian mesh smoothing algorithm is used. Alternatively, if you have a relatively small model (less than 1000 nodes in the mesh smooth area), you can select a local gradient mesh smoothing algorithm. In each iteration the local gradient mesh smoothing algorithm identifies the elements with the worst element quality and improves them by displacing the nodes. Local gradient mesh smoothing usually generates elements having the optimal shape, where the optimal is defined as the ratio of the element volume (area for shell elements) to the corresponding power of its diameter. For larger models the local gradient mesh smoothing algorithm tends to stop before the optimal mesh quality is reached because the computation time becomes excessive. When the mesh smoothing ends prematurely, only the elements with the worst element quality will be smoothed.

Sizing Optimization

Similar to shape optimization, you use sizing optimization at the end of the design process when the general layout of a component is fixed, and only minor changes are allowed by changing the shell thickness in selected regions. A sizing optimization starts with a finite element model that needs minor improvement or with the finite element model generated by a topology optimization. The Optimization module uses a general algorithm for solving sizing optimization problems based on the method of moving asymptotes, which is described in Svanberg (1985).

Typically, the objective of a sizing optimization is to maximize the stiffness of a component while satisfying a weight objective.

A table in *The Available Design Responses for Each Type of Optimization* lists the available design responses for sizing optimization and describes which design responses can be used as objectives and/or constraints. In addition, you can apply geometric restrictions that freeze selected areas and apply member size and symmetry constraints. You can also provide upper and lower bounds for the shell element thickness and group regions into "clusters" of equal shell thickness. You can use the Visualization module to view the variation in shell thickness after a sizing optimization.

An example of using sizing optimization is provided in *Sizing optimization of a gear shift control holder*. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying Clustering to a Sizing Optimization

When you configure a sizing optimization, you can specify that selected regions should contain "clusters" of shell elements of equal thickness. You can use clustering to generate strengthening ribs or rings in the sheet metal structure you are optimizing or to define borders between regions of equal thickness. Clustered regions can be reproduced in manufacturing using sheets of constant thickness; for example, a vehicle "body in white" formed by welding and stamping individual sheet metal structures. To allow for maximum design flexibility,

you should first optimize your structure without specifying clustering and use the initial design to decide which regions to cluster in your final optimization.

Figure 4 shows a sheet metal arm that was optimized with “free” sizing optimization that allows the thickness to be modified in the design area regardless of the thickness of adjacent shell elements. *Figure 5* shows the same model with a clustered ring of shell elements of equal thickness in the design area. The part generated by free optimization is stiffer than the part generated by clustered optimization, but the part generated by free optimization would be impractical to manufacture.

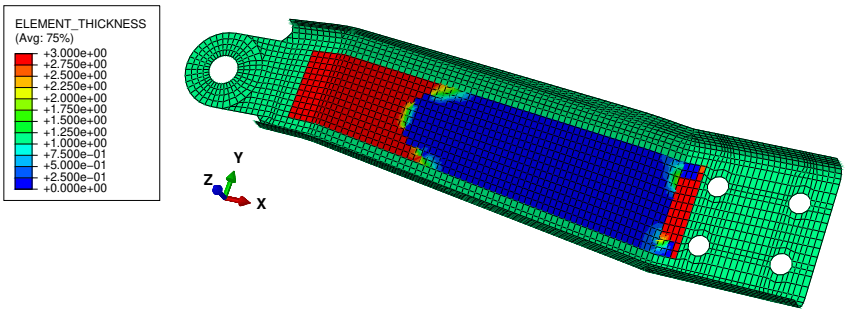


Figure 4: Absolute value of shell thickness after free optimizing.

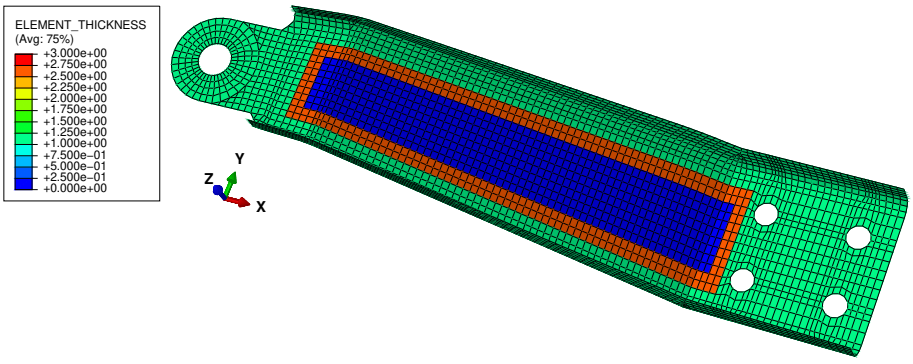


Figure 5: Absolute value of shell thickness after optimizing with clustering.

Bead Optimization

Bead optimization adds stiffening beads to a shell structure to increase the moment of inertia, which leads to a greater stiffness or higher eigenfrequencies. The resulting beads are easy to reproduce in a sheet metal stamping process and add no mass and little cost to the finished product.

Figure 6 shows the von Mises stress in a plate that is fixed at the edges and has a pressure load applied to the center. The image on the left shows the original configuration. The image on the right shows how condition-based

bead optimization has introduced circular stiffening beads and significantly reduced the stress at the center of the plate.

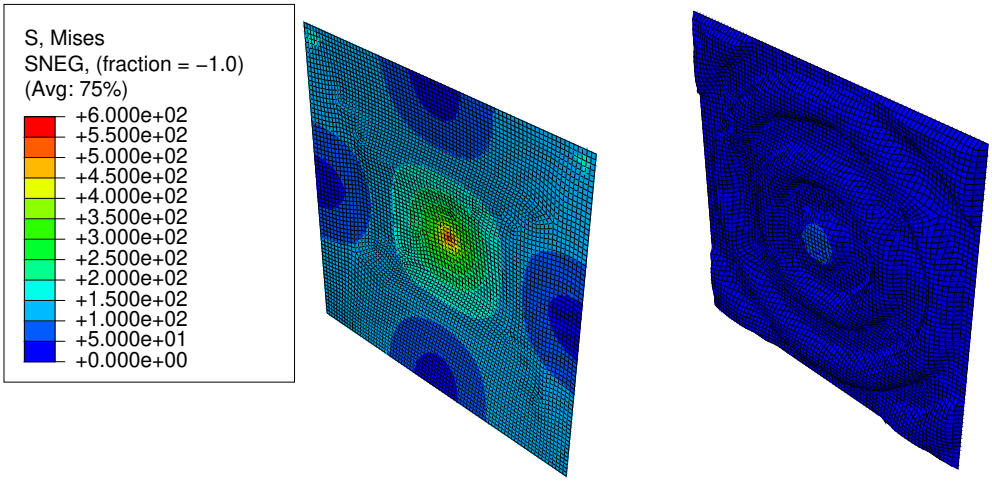


Figure 6: The effect of condition-based bead optimization.

Typically, the objective of a bead optimization is to maximize the stiffness of a component or to minimize the displacement of critical nodes. Bead optimization does not always result in improved mechanical behavior. The following conditions should be true for the design problem to be suited to bead optimization:

- the design area should be subjected to bending or a bending mode, and
- the design area should not be in a predominately membrane stress state. The addition of a bead in this case may make the structure softer.

An example of using bead optimization is provided in *Bead optimization of a plate*. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying Mesh Smoothing to a Condition-Based Bead Optimization

During a bead optimization, the Optimization module moves the nodes of the shell elements in the direction of the shell normal. To maintain the quality of the shell elements during a condition-based bead optimization, the Optimization module can apply mesh smoothing to the design area to limit the displacement of the nodes. Two forms of mesh smoothing are provided—node smoothing and element smoothing. Node smoothing prevents large displacements by limiting the displacement of the nodes at the boundary between the design area and the rest of the model or where active design variable constraints are restraining the displacement of nodes. Element smoothing prevents nodes from overlapping in regions of high curvature where the bead is being formed. You must have a good quality finite element mesh before you start the bead optimization, especially in areas where you expect beads to be formed.

General Versus Condition-Based Bead Optimization

Bead optimization supports two algorithms—the general algorithm, which is more flexible and can be applied to most problems but does not generate a distinct bead pattern, and the condition-based algorithm, which is more efficient at creating beads but has limited capabilities. By default, the Optimization module uses the condition-based algorithm; however, you can choose which algorithm to use when you create the optimization task. Each algorithm has a different approach for determining the optimized solution.

Figure 7 shows the plate model optimized with general bead optimization. While the stresses have been similarly reduced, the seemingly random bead structure that is generated would be hard to manufacture, unlike the regular circular beads generated by the condition-based bead optimization.

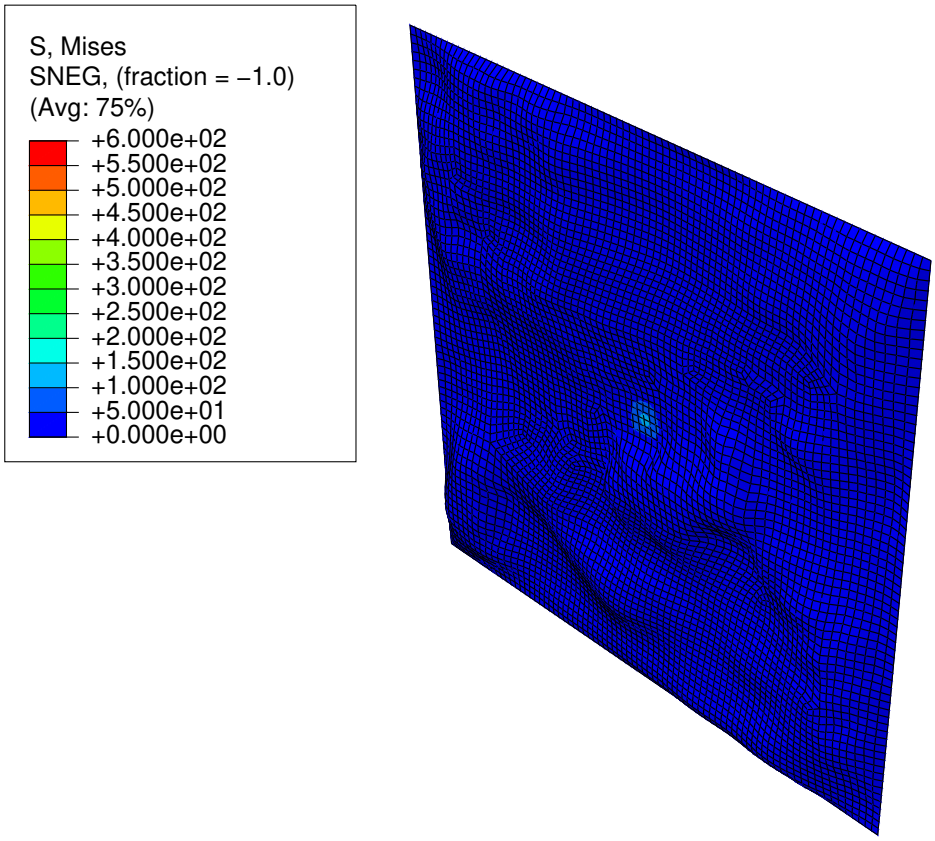


Figure 7: The effect of general bead optimization.

Algorithms

General, or sensitivity-based, bead optimization uses an algorithm that makes it possible to define very complex optimization tasks. It has been shown in industrial size examples that the method is very powerful and attractive, especially for dynamic problems. The algorithm does not use a bead filter, and the results of the optimization are not necessarily a distinct bead pattern. In contrast, condition-based bead optimization uses a special bending hypothesis developed at the Karlsruhe University, Institute of Product Engineering (IPEK) and described in Emmrich (2004). The algorithm determines the orientation of the maximum bending stress for each point in the design and uses filters to generate beads along the bending trajectories.

Controlling the Bead Height and Width

For a condition-based bead optimization, you directly limit the bead height by creating a bead height design response that is referenced by a constraint. You can enter a value for the bead width, or you can allow Abaqus/CAE to calculate the width from the following:

$$\text{bead width} = 2 \times \max(2 \times \text{height of bead}, 3.2 \times \text{average element edge length})$$

Typically, the bead height is small in comparison to the element size, in which case the bead width is a function of the average element length. Otherwise, if the bead height is large in comparison to the element size, the bead width is a function of the bead height.

For a general bead optimization, you must limit the bead height by creating geometric restrictions that limit the displacement in the growth and shrink directions. The growth and shrink geometric restrictions are required

because they are the only restrictions on the displacement of the nodes. You cannot control the bead width for a general bead optimization.

Number of Optimization Design Cycles

The number of design cycles used by the general algorithm is unknown before the optimization starts, but the general algorithm normally requires about 20 design cycles to converge on a solution. The condition-based algorithm is more efficient and always uses three design cycles.

Analysis Types

The general algorithm supports the responses of linear static, linear eigenfrequency, and frequency response finite element analyses. The condition-based algorithm is recommended only for static analyses; however, the algorithm supports all the Abaqus analysis types that generate a stress tensor.

Objective Functions and Constraints

The general bead optimization algorithm can use one objective function and several constraints, where the constraints are all inequality constraints. A variety of design responses can be used to define the objective and the constraints, such as strain energy, displacements and rotations, reaction and internal forces, eigenfrequencies, and moment of inertia. The condition-based bead optimization algorithm is more efficient; however, it is less flexible and supports only strain energy (a measure of stiffness) or eigenfrequency as the objective function and the bead height as an equality constraint.

Tables in *The Available Design Responses for Each Type of Optimization* list the available design responses for general and condition-based bead optimization and describe which design responses can be used as objectives and/or constraints. In addition, you can apply geometric restrictions that freeze selected areas and apply bead size and symmetry constraints.

Optimization Models

In this section:

- *Design Responses*
- *Objectives and Constraints*
- *Creating Abaqus Optimization Models*

Design Responses

Products: Abaqus/CAE

References:

- *About Structural Optimization*
- *Configuring design responses*

Overview

A design response:

- is a single scalar value, such as the volume of your structure;
- is calculated by the Optimization module by reading results and model data from the output database file;
- can be computed by Abaqus for a general topology optimization to improve performance;
- can be referred to from objective functions and constraints (for example, you can create an objective function that tries to minimize the displacement at a node or a constraint that forces the weight of the structure to be reduced by at least 50%); and
- is available only for certain analysis procedures (for example, you must perform an eigenvalue extraction analysis if you select a design response that tries to maximize the lowest eigenfrequencies).

Design Responses

The Optimization module provides a number of different design responses.

Bead Height

The bead height has the most significant effect on the stiffness of a shell structure that is being optimized with bead optimization. Usually, the greater the bead height the greater the stiffness, although manufacturing restrictions will limit how deep you can draw a bead. When the optimization starts, the bead height is initially zero. As the optimization proceeds, it moves the nodes in the shell normal direction until they reach the bead height. A bead height design response is available only as a constraint in a condition-based bead optimization.

Abaqus/CAE Usage: Optimization module: **Task->*condition-based bead task***, **Design Response->Create**: **Single-term**, **Variable: Bead height**

Center of Gravity

You can use the center of gravity of a selected region as a design response in an optimization. You can choose the center of gravity in the three principal directions:

$$x_g = \frac{\int \rho x dV}{\int \rho dV},$$

$$y_g = \frac{\int \rho y dV}{\int \rho dV}, \text{and}$$

$$z_g = \frac{\int \rho z dV}{\int \rho dV}$$

When the Optimization module calculates the center of gravity, the elements are scaled with the current relative density defined in your Abaqus model.

For example, you might want to constrain the center of gravity in the *Y*-direction so that it remains within a minimum and maximum range during the optimization. The design response can consider the center of gravity of the whole model or a region of the model.

If you select a local coordinate system, the Optimization module uses both the direction of the axes and the position of the origin to recalculate the center of gravity. The Optimization module applies the global coordinate system if you do not select a local coordinate system.

When the Optimization module calculates the center of gravity, it treats shell and membrane regions as three-dimensional regions by applying the thickness of the region. The Optimization module calculates the center of gravity using only the element types that are supported by topology optimization. As a result, the center of gravity calculated by the Optimization module might not be the same as the center of gravity calculated by Abaqus/Standard or Abaqus/Explicit; for example, if your model contains wire regions.

Abaqus/CAE Usage: Optimization module: **Task->*general topology or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Center of gravity**

Damage

Typically, you use shape optimization to modify the surface geometry of a component to minimize stress concentrations. In most cases reducing the stress levels leads to a significant increase in durability. However, it is possible that the regions of peak stress identified by a static analysis differ from the regions of maximum damage identified from a durability (or damage) analysis, and using shape optimization alone to modify the surface geometry may decrease the durability. To avoid this situation, you can incorporate a durability solver in the optimization loop to ensure that you are both reducing stress levels and increasing durability.

You can create a design response that uses damage only in a shape optimization, and it can be used only as an objective function—minimization of the maximum damage in the critical areas. The Optimization module provides damage values as equivalent stress values. You can create a damage design response where the durability analysis runs only once in the optimization loop. Alternatively, you can create a damage design response when the durability analysis contains load case information and runs multiple times in the optimization loop.

Abaqus/CAE Usage: Optimization module: **Task->*shape task***, **Design Response->Create**: **Single-term**, **Variable: Damage (single)**

Optimization module: **Task->*shape task***, **Design Response->Create**: **Single-term**, **Variable: Damage (multiple)**

Displacement and Rotation

In most optimization problems you will use displacement and/or rotation to define your objective function or constraints. For example, the maximum displacement of a vertex could be either an objective or a constraint of an optimization. The performance of the optimization is improved if you apply displacements and rotations to only vertices or to small regions. In addition, performance is improved if you assign regions that are used to define displacements or reactions as frozen regions (the Optimization module will not remove elements from frozen regions during the optimization).

Table 1 lists the available displacement and rotation variables.

Table 1: Displacement and rotation variables for a general and sizing topology optimization.

	Displacement	Rotation
i-direction	u_i	θ_i
Absolute	$\sqrt{u_i^2 + u_j^2 + u_k^2}$	$\sqrt{\theta_i^2 + \theta_j^2 + \theta_k^2}$
Absolute in *i*-direction	$\sqrt{u_i^2}$	$\sqrt{\theta_i^2}$

Abaqus/CAE Usage: Optimization module: **Task->*general topology, general bead, or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Displacement**

Energy Stiffness Measure

The energy stiffness is a measure that has no physical meaning but can be used as an objective function or a constraint in an optimization to optimize the stiffness of a structure that is subjected to both external loading and prescribed displacements.

To optimize the stiffness of a structure with only external loading, the strain energy should be minimized:

$$\mathbf{min}\left(\frac{P \cdot u}{2}\right),$$

where $\boldsymbol{P}$ is the external loading and $\boldsymbol{u}$ is the resulting deflection of the loaded nodes. If only external loading is present, the energy stiffness measure is equal to the total strain energy, also called the compliance.

In contrast, if a load case is driven by prescribed displacements, the elastic energy, or compliance, will decrease only if the structure is made softer. To optimize a structure with only prescribed displacements, the strain energy should be maximized:

$$\mathbf{max}\left(\frac{R \cdot u^*}{2}\right),$$

where $\boldsymbol{u}$ is the prescribed displacement at the nodes and $\boldsymbol{R}$ is the resulting reaction force at the displaced nodes. If only prescribed displacements are present, the energy stiffness measure is equal to the negative of the total strain energy.

The strain energy with both external loads and prescribed displacements is given as

$$\theta = \frac{P \cdot u}{2} + \frac{R \cdot u^*}{2}.$$

The energy stiffness measure is used only for optimization (it has no physical meaning) and is given as

$$\mathbf{min}\left(\frac{P \cdot u}{2} - \frac{R \cdot u^*}{2}\right).$$

You cannot use the energy stiffness measure as a design response in a model that is experiencing thermal loading or mass-dependent loading, such as gravity. An energy stiffness design response must be applied to the entire model. When energy stiffness is used as an objective function, you must select a target that tries to minimize the sum of the weighted differences between the design response and the reference value, regardless of whether an external load and/or prescribed displacements are being applied to the structure.

Abaqus/CAE Usage: Optimization module: **Task->*general topology or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Energy stiffness measure**

Equivalent Strain

If your model is undergoing large deformations, a measure of the stress is not always a good indicator of the model's response. For example, a structure undergoing plastic deformation will, for an ideal plastic material, experience a large constant stress over the plastic area. In these circumstances a measure of the strain is a more reliable indicator of the model's response. You can choose from the following equivalent strains:

- Equivalent plastic (PEEQ, calculated by Abaqus)
- Elastic
- Plastic (calculated by the Optimization module)
- Total (the sum of the elastic and plastic)

Abaqus/CAE Usage: Optimization module: **Task->*shape task***, **Design Response->Create**: **Single-term**, **Variable: Strain**

Equivalent Stresses

Equivalent stresses are the most commonly used objective function of a shape optimization. All of the stress values that are calculated by the Optimization module, whether nodal or from Gauss points or elements, are interpolated to the nodes. For example, you can try to optimize your model with an objective function that tries to minimize the maximum von Mises stresses in a region with stress concentrations or tries to minimize contact pressure in a region with contact. The Optimization module considers only the maximum value of an equivalent stress within a region. The Optimization module issues warnings for nodes that do not have the appropriate stress values. For example, if you select an objective response of contact stress, the Optimization module issues warnings about nodes of elements that are not in contact. If your Abaqus model contains multiple load cases, the design response is evaluated by summing the stress values from each load case.

You can choose from the following equivalent stresses:

- von Mises
- Maximum principal and absolute maximum principal
- Minimum principal and absolute minimum principal
- Second principal
- Beltrami
- Drucker Prager
- Galilei
- Kuhn
- Mariotte
- Mises
- Sandel
- Sauter

- Tresca

You can choose from the following equivalent contact stresses:

- Contact stress pressure
- Total shear contact stress
- Shear contact stress in the 1-direction
- Shear contact stress in the 2-direction
- Total contact stress

Abaqus/CAE Usage: Optimization module: **Task->*shape task or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Stress** or **Contact stress**

Internal Forces and Moments

You can use nodal internal forces and moments of the whole model or a region of the model as an objective function or a constraint in a general topology or sizing optimization.

Table 2 lists the available nodal internal force and moment variables.

Table 2: Nodal internal force and moment variables for a general and sizing topology optimization for the elements e attached to the nodes i.

	Nodal internal force	Nodal internal moment
i-direction	$\sum K_e u_i$	$\sum K_e u_i$
Absolute	$\lvert\sum K_e u_i\rvert$	$\lvert\sum K_e u_i\rvert$
Absolute in i-direction	$\sqrt{u_i^2}$	$\sqrt{u_i^2}$

You must use the global coordinate system with an absolute internal force or with absolute internal moment. Your structure must have stiffness in the direction of the force used in the optimization; otherwise, the internal force will be zero in this direction.

Abaqus/CAE Usage: Optimization module: **Task->*general topology, general bead, or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Internal force** or **Internal moment**

Modal Eigenfrequency Analysis

Modal eigenvalues are the simplest dynamic responses in structural analysis. Typical uses of data from an eigenfrequency analysis during a topology optimization include the following:

- maximize the lowest eigenfrequencies,
- maximize a selected eigenfrequency,
- constrain an eigenfrequency to be higher or lower than a given value,
- maximize or minimize an eigenfrequency at a certain mode, and
- perform a bandgap optimization that force modes away from a certain frequency.

The Optimization module supports two approaches for optimizing the eigenfrequencies:

- single eigenfrequencies from modal analysis and

- the Kreisselmeier-Steinhauser formulation.

The Kreisselmeier-Steinhauser formulation is the more efficient of the two approaches and should be used whenever possible. The only advantage of optimizing single eigenfrequencies is that you can use the sum of the eigenfrequencies as a constraint in a general topology or sizing optimization, which you cannot do with the Kreisselmeier-Steinhauser formulation.

When you are trying to maximize the lowest eigenfrequency, it is recommended that you consider not only the first eigenfrequency but also at least the next two highest natural frequencies. During the optimization, the various natural frequencies are weighted by their distance from the lowest natural frequency—the closer a natural frequency approaches the first natural frequency during the optimization, the more it is weighted. You should use the Kreisselmeier-Steinhauser eigenvalue formulation if you are trying to maximize the lowest eigenfrequency or, in particular, if you are trying to maximize more than one of the lowest eigenfrequencies. You do not need to use mode tracking if you are using the Kreisselmeier-Steinhauser formulation to maximize the lowest eigenfrequency, but you should use mode tracking for the higher modes because the modes might switch. For example, while the model is being optimized, the frequency of the first mode is maximized and the second eigenmode can become the mode with the lowest frequency.

Abaqus/CAE Usage: Optimization module: **Task->*general topology, general bead, or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Eigenfrequency from modal analysis** or **Eigenfrequency calculated with Kreisselmeier-Steinhauser formula**

Moment of Inertia

You can use a moment of inertia design response in an optimization to minimize the rotational inertia about a selected axis. You can use the sum of the moment of inertia of the whole model or a region of the model as an objective function or a constraint in a general topology or sizing optimization.

You can choose the moment of inertia in the three principal directions or the three principal planes:

$$I_x = \int \rho \left(y^2 + z^2\right) dV,$$

$$I_y = \int \rho \left(x^2 + z^2\right) dV,$$

$$I_z = \int \rho \left(x^2 + y^2\right) dV,$$

$$I_{xy} = I_{yx} = -\int \rho xy dV,$$

$$I_{xz} = I_{xz} = -\int \rho xz dV,$$

$$I_{yz} = I_{yz} = -\int \rho yz dV.$$

If you select a local coordinate system, the Optimization module uses the direction of the axes to recalculate the center of gravity. The Optimization module applies the global coordinate system if you do not select a local coordinate system.

When the Optimization module calculates the moment of inertia, it treats shell and membrane regions as three-dimensional regions by applying the thickness of the region. The Optimization module calculates the moment of inertia using only the element types that are supported by topology optimization. As a result, the moment of inertia calculated by the Optimization module might not be the same as the moment of inertia calculated by Abaqus/Standard or Abaqus/Explicit; for example, if your model contains beam or truss elements (wire regions in Abaqus/CAE).

The moment of inertia with respect to any two orthogonal axes is zero if you have selected either of the axes to be an axis of symmetry.

Abaqus/CAE Usage: Optimization module: **Task->*general topology, general bead, or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Moment of inertia**

Nodal Strain Energy Density

The nodal strain energy density, $u = \sigma_{ij}\varepsilon_{ij}$, is a local point-wise strain energy that can provide a better representation of failure than stresses in nonlinear materials.

Abaqus/CAE Usage: Optimization module: **Task->*shape task***, **Design Response->Create**: **Single-term**, **Variable: Strain energy density**

Reaction Forces and Moments

Nodal reaction forces and moments can be used as a design response only in general and sizing topology optimization. As with displacements, the performance of the optimization is improved if you apply reaction forces or moments to only vertices or to small regions and assign those regions as frozen regions (the Optimization module will not remove elements during the optimization).

Table 3 lists the available nodal reaction force and moment variables.

Table 3: Nodal reaction force and moment variables for a general and sizing topology optimization for the elements e attached to the nodes i.

	Nodal reaction force	Nodal reaction moment
i-direction	$\sum K_e u_i$	$\sum K_e u_i$
Absolute	$\lvert\sum K_e u_i\rvert$	$\lvert\sum K_e u_i\rvert$
Absolute in i-direction	$\sqrt{u_i^2}$	$\sqrt{u_i^2}$

You cannot use a reference coordinate system with an absolute reaction force or with an absolute reaction moment. Your structure must have stiffness in the direction of the force used in the optimization; otherwise, the reaction force will be zero in this direction.

Abaqus/CAE Usage: Optimization module: **Task->*general topology, general bead, and sizing task***, **Design Response->Create**: **Single-term**, **Variable: Reaction force** or **Reaction moment**

Scaled Centroidal Von Mises Stress

You can use the scaled element centroidal von Mises stress of the whole model or a region of the model as an objective function in an optimization. You must avoid regions with stress singularities caused by external loads or boundary conditions.

The scaled element centroidal von Mises stress is defined as

$$= \mathbf{Max}\left|\frac{(\sigma_{vMises})^2}{(f(\rho_i)\sigma_y)^2}\cdot\sigma_y\right|,$$

where σ_{vMises} is the element centroidal von Mises stress, σ_y is the reference stress, and ρ_i is a factor for interpolating the stresses of elements that have a reduced current relative density because of the topology optimization. The weighting factor and the interpolation are required for convergence during the optimization.

The von Mises stress is calculated at the centroid of the element to avoid stress singularities that might be present in the initial model or might appear in an optimized structure before it is smoothed. You cannot compare the scaled element centroidal von Mises stress with the von Mises stress calculated by Abaqus. The two measures are equal only when the element is solid and has a relative density of 1.0.

You can provide the reference stress when you create the objective function, or the Optimization module can calculate the reference stress during the initial optimization iteration. If you provide the reference stress, the value should not be too low or numerical singularities will result. The reference stress is given as

$$\sigma_y = \mathrm{Min}\left\{\alpha_1 \,\mathrm{Max}\left|\frac{(\sigma_{vMises})^2}{\left(f(\rho_i)\sigma_{ref_1}\right)}.\sigma_{ref_1}\right| + \alpha_2 \,\mathrm{Max}\left|\frac{(\sigma_{vMises})^2}{\left(f(\rho_i)\sigma_{ref_2}\right)^2}.\sigma_{ref_2}\right|\right\}.$$

You can define multiple load cases for the scaled element centroidal von Mises stress measure. Static linear analysis is supported. Static nonlinear analysis supports only contact nonlinearities. Nonlinear materials and geometrical nonlinearities, such as large deformations, are not supported.

Abaqus/CAE Usage: Optimization module: **Task**->***general topology task***, **Design Response**->**Create**: **Single-term**, **Variable: Stress**

Strain Energy

The compliance of a structure is a measure of its overall flexibility or stiffness and is defined as the sum of the strain energy of all the elements, $\sum u^t k u$ for linear models, where u is the displacement vector and k is the global stiffness matrix. Compliance is the reciprocal of stiffness, and minimizing the compliance is equivalent to maximizing the global stiffness. If a load case is driven by forces or pressures, you should choose to minimize the strain energy to maximize the global stiffness. However, if a load case is driven by a thermal field, strain energy decreases when the optimization modifies the structure to make it softer. As a result, you should always choose to maximize the strain energy because attempting to minimize the strain energy can result in a stiff structure. In addition, you should always choose to maximize the strain energy if prescribed displacements are applied to your model.

Topology optimization considers the total strain energy for all of the elements; therefore, if you choose strain energy as an objective function, you must apply the objective to the entire model. You cannot use strain energy as a constraint in your optimization.

Abaqus/CAE Usage: Optimization module: **Task**->***condition-based topology or general bead task***, **Design Response**->**Create**: **Single-term**, **Variable Strain energy**

Volume

The volume is defined as the sum of the volume of the elements in the design area, $\sum V_e$, where V_e is the element volume. During a topology optimization, the elements are scaled with the current relative density defined in your Abaqus model. For most optimization problems, you must apply a volume constraint. For example, if you are trying to minimize the strain energy (maximize the stiffness) and do not apply a volume constraint, the Optimization module simply fills the entire design area with material.

Abaqus/CAE Usage: Optimization module: **Task**->***condition-based topology task***, **Design Response**->**Create**: **Single-term**, **Variable: Volume**

Weight

The weight is defined as the sum of the weight of all the elements in the design area, $\sum W_e$, where W_e is the element weight. The Optimization module scales elements using the current relative density. For most optimization problems, you must apply either a volume or a weight constraint. Using weight instead of volume allows you to constrain the optimized model to a specified physical weight and accounts for regions composed of materials with different densities. The Optimization module uses only supported element types when calculating the weight.

Abaqus/CAE Usage: Optimization module: **Task->*general topology or sizing task***, **Design Response->Create**: **Single-term**, **Variable: Weight**

Design Response Operators

You must specify the operation that the Optimization module will use to arrive at a single scalar value for the design response, although some restrictions apply. For example, a volume design response can only use the sum of the volume within the design area. A design response that calculates the von Mises stress must use the maximum value of the stress within a region of the model. (None of the operators are relevant when the Optimization module calculates a dynamic frequency design response.) The following design response operators are provided by the Optimization module:

Minimum or maximum

The minimum or maximum value within the selected region. The Optimization module allows only the maximum operator for stress, contact stress, and strain design responses.

Sum

The sum of all the values within the selected area. The Optimization module allows only the sum operator for volume, weight, moment of inertia, and gravity design responses.

The Available Design Responses for Each Type of Optimization

The design responses you can create are dependent on the type of structural optimization you are performing—topology, shape, sizing, or bead. The type of optimization also controls whether you can use a design response as an objective, as a constraint, or as both.

Table 4: Design responses for condition-based topology optimization.

Design response	Objective	Constraint
Strain energy	√	
Volume		√

Table 5: Design responses for general topology optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmeier-Steinhauser formula	√	

Design response	Objective	Constraint
Eigenfrequency from modal analysis	√	√
Energy stiffness measure	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√
Scaled centroidal von Mises stress	√	√
Strain energy	√	
Volume		√
Weight	√	√

Table 6: Design responses for shape optimization.

Design response	Objective	Constraint
Contact stress	√	
Damage	√	
Eigenfrequency from Kreisselmeier-Steinhauser formula	√	
Equivalent strain	√	
Equivalent stress	√	
Nodal strain energy density	√	
Volume		√
Weight		√

Table 7: Design responses for sizing optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmeier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	√
Energy stiffness measure	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√

Design response	Objective	Constraint
Strain energy	√	
Volume		√
Weight	√	√

Table 8: Design responses for general bead optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmeier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√
Strain energy	√	

Table 9: Design responses for condition-based bead optimization.

Design response	Objective	Constraint
Bead height		√
Eigenfrequency from Kreisselmeier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	
Strain energy	√	

Operating on Design Responses

You can define a design response that is a combination of the single values generated by multiple design responses; for example, you can add values or find the maximum of several values. You can also define a design response that is the result of an operation on another design response; for example, the difference between the value of the design response at different nodes.

For example, you can create two design responses that correspond to the displacement in the 1-direction of two selected vertices. Alternatively, you can create a design response that is the difference between the displacement in the 1-direction of two selected vertices. You can then define a constraint that forces the design response to be close to zero. In effect, the constraint forces the two selected vertices to move together in the 1-direction.

Abaqus/CAE Usage: Optimization module: **Design Response->Create**: **Combined-term**

References

- Bakhtiary, N., P. Allinger, M. Friedrich, F. Mulfinger, J. Sauter, O. Müller, and J. Puchinger, "A New Approach for Size, Shape and Topology Optimization," SAE International Congress and Exposition, Detroit, Michigan, USA, February 26–29, 1996.
- Bendsøe, M. P., E. Lund, N. Ohloff, and O. Sigmund, "Topology Optimization - Broadening the Areas of Application," *Control and Cybernetics*, vol. 34, pp. 7–35, 2005.
- Bendsøe, M. P., and O. Sigmund, Topology Optimization: Theory, Methods and Applications, Springer-Verlag, Berlin Heidelberg New York, 2003.
- Bendsøe, M. P., and O. Sigmund, "Material Interpolations in Topology Optimization," *Archive of Applied Mechanics*, vol. 69, pp. 635–654, 1999.
- Clausen, P. M., and C. B. W. Pedersen, Non-Parametric Large Scale Structural Optimization, ECCM 2006 III European Conference on Computational Mechanics, Lisbon, Portugal, June 5–9, 2006.
- Cook, R. D., D. S. Malkus, and M. E. Plesha, Concepts and Applications of Finite Element Analysis, John Wiley & Sons Inc., 1989.
- Emmrich, D., "Entwicklung einer FEM-basierten Methode zur Gesaltung von Sicken für biegebeanspruchte Leitstützstrukturen im Konstruktionsprozess," *Forschungsberichte des Instituts für Produktentwicklung, 13. Karlsruhe*, 2004.
- Hansen, L. V., "Topology Optimization of Free Vibrations of Fiber Laser Packages," *Structural and Multidisciplinary Optimization*, vol. 29(5), pp. 341–348, 2005.
- Olhoff, N., and J. Du, Topology Optimization of Vibrating Bi-Material Plate Structures with Respect to Sound Radiation, IUTUAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives, M. P. Bendsøe, N. Olhoff, and O. Sigmund, eds., pp. 147–156, Springer, 2006.
- Pedersen, C. B. W., and P. Allinger, Industrial Implementation and Applications of Topology Optimization and Future Needs, IUTUAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives, M. P. Bendsøe, N. Olhoff, and O. Sigmund, eds., pp. 147–156, Springer, 2006.
- Sigmund, O., and J. S. Jensen, "Systematic Design of Phononic Band Gap Materials and Structures by Topology Optimization," *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences*, vol. 361, pp. 1001–1019, 2003.
- Stolpe, M., and K. Svanberg, "An Alternative Interpolation Scheme for Minimum Compliance Optimization," *Structural and Multidisciplinary Optimization*, vol. 22, pp. 116–124, 2001.
- Svanberg, K., "The Method of Moving Asymptotes—A New Method for Structural Optimization," *International Journal for Numerical Methods in Engineering*, vol. 24, pp. 359–373, 1987.

Objectives and Constraints

Products: Abaqus/CAE

References:

- *About Structural Optimization*
- *Creating objective functions*
- *Creating constraints*
- *Configuring geometric restrictions*
- *Creating local stop conditions*

Overview

For an optimization problem:

- an objective function defines the objective of the optimization;
- a constraint imposes limitations on the optimization and defines a feasible design;
- geometric restrictions impose limitations on the topology or shape of the structure that can be generated by the optimization; and
- stop conditions define when an optimization task is considered complete.

Objective Functions

Objective functions define the objective of the optimization. An objective function is a single scalar value that is formulated from a set of design responses. For example, if the design responses are defined from the strain energy of the nodes in a region, the objective function could minimize the sum of the design responses; i.e., minimize the sum of the strain energy, in effect maximizing the stiffness of the region.

An optimization problem can be stated as:

$$\mathbf{min}(\Phi(U(x), x)),$$

where Φ is the objective function that depends on the state variables, U, and the design variables, x.

The formula for the objective function that tries to minimize N design responses can be stated as:

$$\Phi_{\mathbf{min}} = \mathbf{min}\left(\sum_{i=1}^{N} W_i\left(\varphi_i - \varphi_i^{ref}\right)\right),$$

where each design response, φ_i, is given a weight, W_i, and a reference value, φ_i^{ref}. The formula for the objective function that tries to maximize N design responses can be stated as:

$$\Phi_{\mathbf{max}} = \mathbf{max}\left(\sum_{i=1}^{N} W_i\left(\varphi_i - \varphi_i^{ref}\right)\right).$$

The default weighting factor is 1.0. For a topology optimization the default reference value is 0.0; for shape and sizing optimization the default reference value is calculated by the Optimization module. For the most common optimization problems you do not need to change the default values of the weighting factor and the reference

value. However, in some cases you may have to change the weighting factor to balance the effect of an objective function that is dominating the optimization. You should be aware that changing the weighting factor can have a significant impact on the final design. In addition, a design response that is dominant at the start of the optimization may have less effect as the Optimization module modifies your model.

An objective function that tries to minimize the maximum design response is an important optimization formulation. During each design cycle the Optimization module first determines which of the set of weighted design responses has the maximum value and then tries to minimize that design response. In many problems, minimizing the maximum design response provides a satisfactory solution because it reduces the maximum of a number of design responses. For example, if your design responses are defined from the stress in multiple regions of your model, minimizing the maximum design response attempts to minimize the stress in the region that is exhibiting the maximum stress. The formula can be stated as:

$$\Phi_{minmax} = \min\left(\max_i \left\{W_i \left(\varphi_i - \varphi_i^{ref}\right)\right\}\right).$$

The design responses provided with the Optimization module are listed in *Design Responses*.

Defining the Target of an Objective Function

The target of an objective function can be minimized or maximized. Alternatively, the target of an objective function can be set to minimize the maximum, such that the design response targets the maximum value, and the objective attempts to minimize that maximum value. In all cases, the weighting and reference values of the design responses are accounted for.

Abaqus/CAE Usage: Optimization module: **Objective Function**->**Create**: **Target**

Constraints

As outlined in the previous section, an optimization problem can be stated as:

$$\min(\Phi\left(U\left(x\right), x\right)),$$

where Φ is the objective function that depends on the state variables, U, and the design variables, x. Constraints, Ψ, can be applied to the optimization problem, and constraints, K_i, can be applied to the design variables:

$$\begin{aligned} \Psi_i\left(U\left(x\right), x\right) &\leq 0, \\ K_i\left(x\right) &\leq 0, \end{aligned}$$

where $\Psi_i\left(U\left(x\right), x\right) \leq \Psi_i^*$ and Ψ_i is the design response that is constrained by the value Ψ_i^*. In addition, $K_i\left(x\right) \leq K_i^*$, where K_i is an expression for the layout of the design variables, such as manufacturability, and K_i^* is the constraint on the design variables.

The Optimization module can arrive at a solution that optimizes the objective function; however, if the constraints are not satisfied, the result of the optimization may not be a feasible design. A constraint is based on a design response and, similar to a design response, is formulated from a single scalar value. Most optimizations have constraints that prevent the optimization from arriving at a trivial solution. For example, if you are trying to maximize the stiffness of a structure, the Optimization module will simply fill the entire design area if you do not apply any constraints. However, if you apply a weight constraint that limits the weight to 50% of its initial value, the Optimization module is forced to seek an optimum solution that both optimizes the stiffness objective and satisfies the weight constraint. You can apply only volume constraints to both topology optimization and to shape optimization; you cannot use volume as an objective function. You cannot apply multiple constraints of the same type, such as volume, to the whole model or to a single region.

Abaqus/CAE Usage: Optimization module: **Constraint**->**Create**

Applying Constraints to Regions

You can apply different constraints to different regions of your model. In addition, those regions can have different material properties or a material property can vary within a region. When the Optimization module calculates the design response, it considers varying material properties within the region. You cannot apply multiple volume constraints to the whole model or to a single region.

Geometric Restrictions

Geometric restrictions are constraints that are applied directly to the design variables. Geometric restrictions allow you to model design limitations and manufacturing limitations.

Defining a Frozen Area

You can specify that a region within the optimization region is excluded from the optimization by freezing the region. For example, you could exclude a circular shaft that forms a bearing surface or a boss that is used to attach the structure to a rigid surface. You must freeze regions that are used to apply prescribed conditions. To simplify this operation, you can request that the Optimization module automatically freeze regions that are used to apply prescribed conditions and loads when you create an optimization process.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Frozen area**

Specifying Minimum and Maximum Member Size

In most cases you should try to avoid the generation of thin trusses in the structure by defining a minimum member size. However, the Optimization module cannot ensure that the optimized structure will not contain regions with a diameter that is smaller than the minimum member size. The minimum member size must be larger than the average element edge length. The maximum member size must be larger than twice the element length; otherwise, the optimization algorithm may experience issues with element connectivity. A coarse mesh and a fine mesh lead to an optimization with the topological equivalent result if you specify the same minimum member size for both cases. The Optimization module will not generate a thin region where prescribed conditions have been applied to the structure. Removing material from these regions may result in the structure collapsing.

If your structure will be cast, you may want to avoid the generation of overly thick parts by specifying a region with a maximum member size. The optimization process will avoid creating a thick region by generating several thinner regions. You do not need to specify both a maximum and a minimum member size. The Optimization module assumes the value that you enter for the maximum member size also applies to the minimum member size and will generate trusses of the specified size. The combination of a maximum member size constraint with a restraint that imposes a pull direction, such as a moldable or stampable manufacturing constraint, is allowed only for a general topology optimization. (The “pull direction” is the direction in which the two halves of a mold separate or the direction in which a stamping tool moves.)

Computational time increases significantly when you specify regions with a minimum or maximum member size. Therefore, you should apply the member size restrictions only to regions where thin or thick members must be avoided. You should run an optimization without member size restrictions to identify such regions.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Member size**

Applying Manufacturing Restrictions

The topology optimization process always creates a structural layout that satisfies the objective function and the constraints; however, the design may be impossible to create using standard manufacturing techniques, such as casting and forging. You can apply geometric restrictions that force the topology optimization process to consider only designs that can be manufactured. For example, when you are using topology optimization you can force

the Optimization module to create a castable shape that can be extracted from a mold or a stampable shape that can be created with a tool and die.

Maintaining a Moldable Structure

In cases where bending and torsion loads are applied, topology optimization is likely to generate a model with hollow areas or with undercuts that cannot be manufactured. You can prevent the topology optimization from generating cavities and undercuts by specifying the following:

- A forgeable structure that can be removed from the forging die, as shown in *Figure 1*.

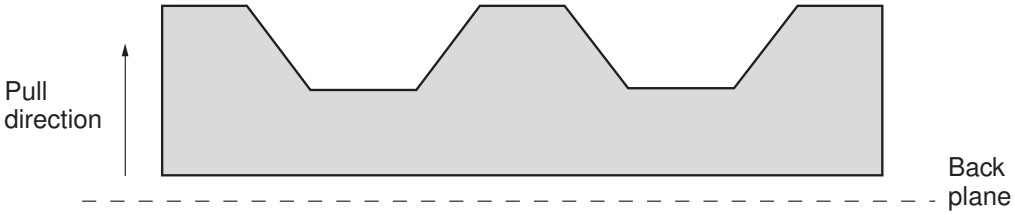


Figure 1: A forgable part.

- A moldable structure that can be removed from two halves of a mold, as shown in *Figure 2*.

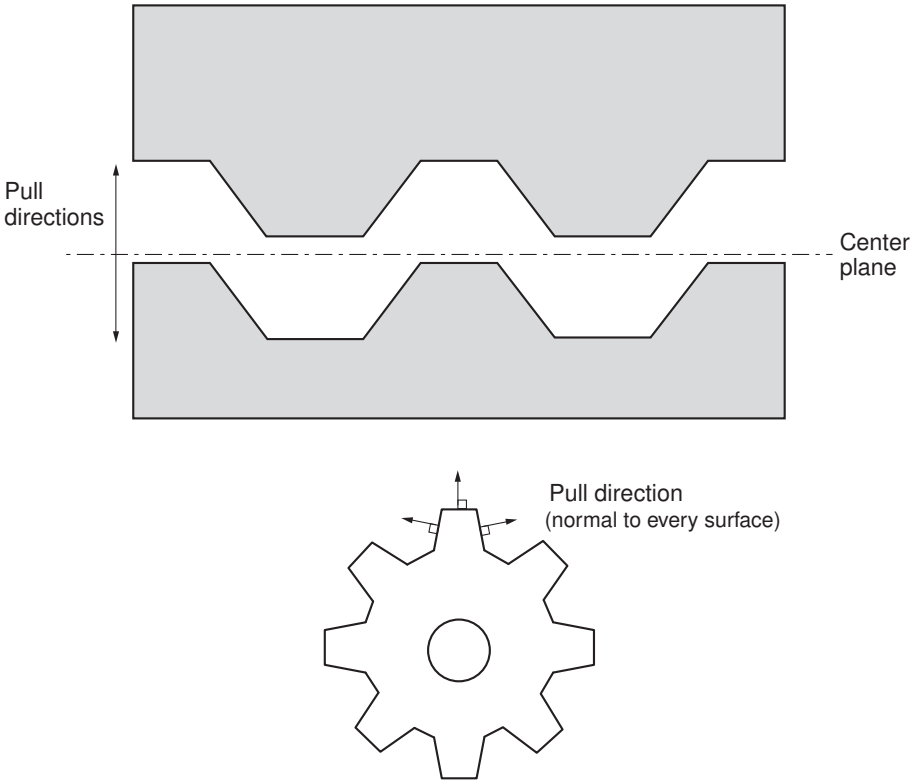


Figure 2: A moldable part.

In contrast, *Figure 3* illustrates parts with a cavity and an undercut that are not moldable.

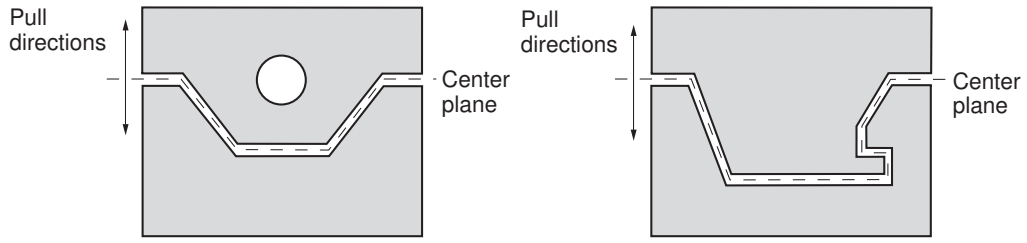


Figure 3: Cavities and undercuts prevent a part from being moldable.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Demold control**; **Demold technique**, **Demolding with a central plane**

Optimization module: **Geometric Restriction->Create**: **Demold control**; **Demold technique**, **Demolding at the region surface**

Optimization module: **Geometric Restriction->Create**: **Demold control**; **Demold technique**, **Forging**

Maintaining a Stampable Structure

You can specify that the structure is to be manufactured by a stamping process. If the optimization process removes one element from the structure, it also removes all elements positioned either behind or in front of the element (with respect to the pull direction), as shown in *Figure 4*.

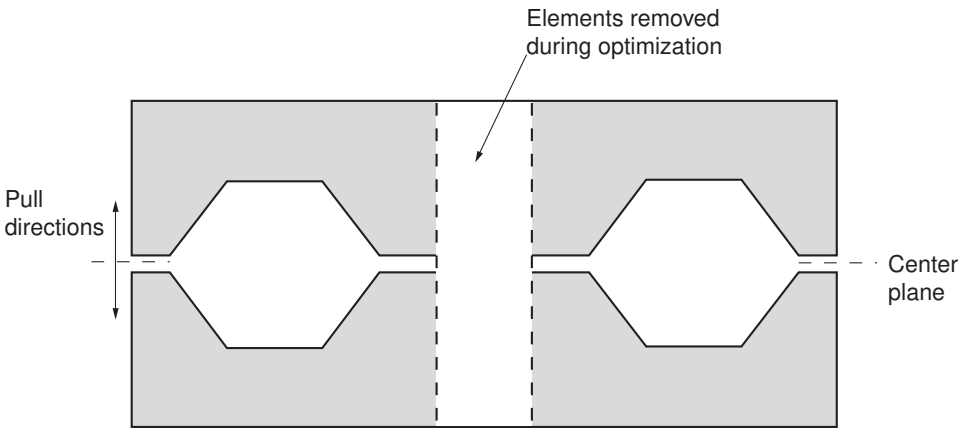


Figure 4: A stampable structure.

The rate at which the Optimization module modifies the element properties should not be set too high if the stamping restriction is activated in a condition-based topology optimization; otherwise, supports or trusses generated by the optimization may become unattached from the rest of the structure.

Abaqus/CAE Usage: Use the following option to create a stamping geometric restriction in a topology optimization:

Optimization module: **Geometric Restriction->Create**: **Demold control**; **Demold technique**, **Stamping**

Use the following option to create a stamping geometric restriction in a shape optimization:

Optimization module: **Geometric Restriction->Create**: **Stamp control**

Use the following option to specify the rate at which the Optimization module modifies the element properties:

Optimization module: **Task->Create**: **Advanced** , **Size of increment for volume modification**

Specifying Bounds on Shell Thickness

You can specify upper and lower bounds on the thickness of shell elements when you are configuring a sizing optimization. The value can be an absolute thickness or a fraction of the initial thickness.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Thickness control**

Specifying a Symmetric Structure

Introducing symmetry constraints into your model can significantly increase the speed at which the Optimization module calculates the optimized structure. You can use the Optimization module to apply the following symmetry constraints:

- Symmetry about an axis or plane (reflection symmetry)
- Symmetry about a point
- Rotational symmetry
- Cyclic symmetry (replication of an area with a given distance)

You can apply a symmetry restriction to unstructured meshes or to tetrahedron meshes in a topology optimization. The elements should be approximately the same size because the resulting symmetry is based on the resolution of the coarsest part of the mesh. In addition, the Optimization module may fail to create the symmetric conditions if the difference in the element size is too large.

To define symmetry for a shape optimization, the Optimization module assembles nodes that are approximately symmetric into a symmetry group (normally there are two symmetric nodes in each symmetry group). The Optimization module then determines the main node of the symmetry group and calculates the design displacements of the client nodes in such a way that they move symmetrically to the plane of the main node.

If you are performing a topology optimization, your meshed Abaqus model does not have to be symmetric before the optimization starts. Conversely, if you are performing a shape optimization, your meshed Abaqus model should be symmetric before the optimization starts to allow the Optimization module to identify symmetric nodes and maintain their symmetry when the surface nodes are moved.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Planar symmetry**, **Point symmetry**, **Rotational symmetry**, or **Cyclic symmetry**

Specifying the Minimum Width

A sizing optimization determines the optimal element thickness when modeling a sheet metal structure with shell elements. Specifying the minimum width of a region containing elements of the same thickness prevents narrow slivers of elements with equal thickness appearing in the solution after a sizing optimization. Specifying a minimum width also prevents oscillations in the shell thickness or a “checkerboard” pattern of element thicknesses.

A coarse mesh and a fine mesh lead to the same optimized solution if the same minimum width is specified in both cases; thus, the optimized solution is independent of the mesh resolution. However, in all cases the minimum width must be larger than the average length of the element edges.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Thickness control**

Specifying Clustering

You can specify that selected regions contain clusters of shell elements of equal thickness after a sizing optimization. You can use clustering to generate strengthening ribs or rings in the sheet metal structure you are optimizing or to define borders between regions of equal thickness. Clustered regions can be reproduced in manufacturing using sheets of constant thickness; for example, a vehicle "body in white" formed by welding and stamping individual sheet metal structures. To allow for maximum design flexibility, you should first optimize your structure without specifying clustering and use the initial design to decide which regions to cluster in your final optimization.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Cluster areas**

Applying Additional Restrictions during a Shape Optimization

Shape optimization determines the displacement of each surface node in an effort to homogenize the stress on the surface and satisfy the objective function and any constraints. The Optimization module does not couple the displacement of neighboring nodes; each of the design nodes can move independently of the other design nodes. For example, during the optimization a planar surface can develop into a nonplanar free-form surface. By coupling the design nodes you can force the optimization to maintain the regularity of a plane.

Coupling conditions restrict the range of solutions for the system and reduce the optimization potential. In addition, defining the appropriate coupling conditions can be very time consuming. To simplify your optimization, you should start with an optimization with as few restrictions as possible and only a few coupling conditions and introduce additional coupling conditions only if they are required.

You can apply additional restrictions while the Optimization module is moving surface nodes during a shape optimization:

- The optimized shape can be manufactured by a tool on a lathe cutting into the model along a specified direction.
- The optimized shape can be manufactured by a tool drilling into the model along a specified direction. The hole created by the tool is symmetric about the axis of the tool. In addition, the tool can be withdrawn from the hole.
- Selected faces in the optimized shape can slide along each other and/or cannot penetrate each other.
- Nodes are restricted to move:
  - along a specified vector,
  - a specified distance either inward or outward (shrinkage or growth),
  - along a specified direction,
  - only along selected degrees of freedom, and
  - only in the direction of applied loads.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction->Create**: **Turn control**, **Drill control**, **Penetration check**, **Slide region control**, or **Vector**

Combining Geometric Constraints

Each geometric constraint that you apply reduces the possibility of Abaqus arriving at an optimized solution. In addition, if you apply too many geometric restrictions, the solution that Abaqus generates may not be the most optimal solution available. Therefore, you should start by allowing Abaqus to perform an optimization with no geometric restrictions applied or with only a limited number. After you have studied the results of the unrestricted, or less-restricted, optimization, you should apply only the restrictions that are required to solve the problem.

You can combine geometric constraints; however, only certain combinations are permissible. Abaqus processes geometrical constraints in the following order:

- Minimum member size
- Symmetry constraints
- Manufacturing constraints
- Maximum member size

Applying one constraint may weaken the effect of another constraint. For example, you cannot define symmetry about a plane in conjunction with a demold pull direction that is not parallel with the axis or plane of symmetry.

The following manufacturing restriction combinations are permissible:

- You can combine symmetry about a plane with a pull direction provided the pull direction is perpendicular or parallel to the plane of symmetry.
- You can combine rotational symmetry with a pull direction provided the pull direction is parallel to the axis of rotation.
- You can combine two symmetries about a plane provided the planes are perpendicular.
- You can specify a minimum member size that is greater than the maximum member size. Abaqus first processes the minimum member size requirement and creates relatively thick supports. The thick supports are subsequently divided into smaller parallel members when Abaqus processes the maximum size requirement.

Stop Conditions

Stop conditions are examined after each design cycle and determine whether an optimization should end because the maximum number of design cycles has been reached or because the optimization has converged on an optimal solution. The Optimization module provides both global and local stop conditions; however, local stop conditions are supported only by shape optimization and are rarely required.

Global Stop Conditions

The global convergence stop condition defines the maximum number of design cycles that should be performed. To limit the number of design cycles, you must define a global stop condition for each optimization task. The default value for the maximum number of design cycles depends on the type of optimization, as shown in *Table 1*.

Table 1: Default maximum number of design cycles.

Optimization Type	Default maximum number of design cycles
Condition-based topology optimization	15
General topology optimization	50
Shape optimization	10
Sizing optimization	50

Optimization Type	Default maximum number of design cycles
Condition-based bead optimization	3
General bead optimization	20

Abaqus/CAE Usage: Job module: **Optimization->Create**: **Maximum cycles**

Local Stop Conditions

Local stop conditions indicate if a shape optimization has converged on an optimal solution. Local stop conditions apply to the displacements or stresses in a region of your model and define when the goals of an optimization have been reached. A local stop condition compares a single scalar value of displacement or equivalent stress to a reference value. The single scalar value can be either the maximum or minimum value over a region or the sum of all the values. The reference value can be taken from the value of the single scalar value after the previous iteration or after the first iteration. In addition, you can modify the reference value by a fixed amount or by a percentage. For example, you can specify a local stop condition that ends the optimization if the sum of the displacements within a region is smaller than 1% of the sum of the displacements after the first optimization cycle. You can define one or two local stop conditions, and you can specify if either or both (default) of the local stop conditions must be met for the Optimization module to end the optimization.

Examples of local stop conditions include the following:

- If you have specified that the displacement or stress should be minimized (or maximized), a local stop condition can end the optimization if the value of the displacement or stress increases (or decreases) after an optimization cycle.
- When the optimization approaches the optimum solution, you can expect only small changes in the value of the displacement or stress. A local stop condition can end the optimization if the relative change in the displacement or stress falls below a tolerance limit after an optimization cycle.
- When the optimization approaches the optimum solution, you can expect only small changes in the sum of the displacements and, therefore, only minor modifications to the model. A local stop condition can end the optimization if the change in the sum of the displacements falls below a tolerance limit after an optimization cycle. You can use the sum of the displacements as a stop condition for optimizations with and without constraints. In addition, this stop condition is suitable for a variety of objective functions, such as stress or frequency.

Abaqus/CAE Usage: Optimization module: **Stop Condition->Create**

Creating Abaqus Optimization Models

Products: Abaqus/CAE

References:

- *About Structural Optimization*
- *Understanding optimization*

Overview

For each design cycle the optimization process:

- generates new material and element properties during topology optimization;
- modifies nodal coordinates during shape optimization;
- sends the modified model to an Abaqus analysis; and
- reads the results of the analysis.

Preparing the Abaqus Model

You should take care to ensure that your Abaqus model is supported by structural optimization. Any restrictions imposed by the use of structural optimization, such as the supported element types, apply only to the design area; regions outside the design area do not play a role in the optimization.

- You must ensure that your Abaqus model can be analyzed and produces the expected mechanical results before you attempt to optimize your model.
- You should account for nonlinearities only if your model is truly nonlinear; the optimization will be significantly less expensive computationally if your Abaqus model is linear. You may want to ensure that an optimization of a linear version of your model produces reasonable results before you introduce geometric or material nonlinearities.
- An optimization takes multiple design cycles to complete, and the time required to reach an optimized solution can be significant. As a result, you must configure your Abaqus model to minimize computational time; for example, by removing small details that are not important to the optimization.
- The Optimization module does not support the use of parts and assemblies in the Abaqus input file. When you run an optimization task, the Optimization module generates a flattened input file that does not use parts and assemblies.
- The Optimization module reads data from the output database (`.odb`) files that are created during each design cycle. The Optimization module requests data only from the end of each step. To minimize the size of the output database files, you should also request data only from the end of each step.

Support for Analysis Types

The following Abaqus analysis types are supported by topology, shape, sizing, and bead optimization:

- Static stress/displacement, general analysis
- Static stress/displacement, linear perturbation analysis
- Extract natural frequencies and modal vectors

Support for Geometric Nonlinearities

You can specify that geometric nonlinearity should be accounted for only during static stress/displacement analyses.

Elements that have limited stiffness, such as elements with hyperelastic material properties, can deform excessively during topology optimization in a nonlinear analysis. This deformation can lead to an adverse effect on the convergence and result in the termination of the analysis. You should be aware of this potential issue when applying topology optimization using hyperelastic materials.

Sizing optimization supports geometric nonlinearity only if the maximum elemental effective total strain for the design elements is less than 2%. Sizing optimization supports geometric nonlinearity outside the design area where any magnitude of total strain for an element is allowed.

Bead optimization does not support geometric nonlinearity.

Support for Multiple Load Cases

If your model is undergoing a sequence of loads, you can significantly reduce the computational cost by defining a multiple load case analysis within a single step.

Support for Multiple Models

A design response can include steps or load cases from multiple Abaqus models. You can incorporate multiple models into your optimization when linear perturbations about a base state are no longer sufficient as load cases. For example, you can simulate nonlinear load cases (which are not supported by Abaqus/CAE) by creating multiple copies of your nonlinear model and by creating a step in each model during which different loads and boundary conditions are applied. For a meaningful optimization, it is expected that each model will have the sameAbaqus/CAE geometry and the same mesh.

Support for Temperature Loading

General topology and sizing optimization support constant temperature loading.

Support for Acceleration Loading

General topology optimization supports prescribed acceleration loading from

- gravity,
- rotational body forces, and
- centrifugal forces.

Coriolis forces are not supported.

Support for Contact during the Optimization

You can avoid contact in optimized regions of your model by defining geometric restrictions, such as casting or minimum member size restrictions. In some cases, you cannot specify the exact boundary conditions early in the design phase. In addition, nonlinear boundary conditions, such as contact definitions, can change if the Optimization module changes the topology of the model.

The optimization process is more efficient if you create an Abaqus model with the appropriate contact definitions and allow Abaqus to calculate the contact. The contact conditions are included in the optimization through the forces at the nodes and the stresses in the elements, and both topology and shape optimization permit contact conditions in the Abaqus model.

You can define a contact surface directly on the edge of the design space in topology optimization. However, if the design edge belongs to a contact surface in shape optimization, you must invert the shape optimization algorithm by entering a negative growth scale factor. You may encounter convergence difficulties in your Abaqus model if you have a complex contact problem or if the optimization results in large changes in the model.

Restrictions on an Abaqus Model Used for Topology Optimization

Topology optimization determines the optimal material distribution in the design space, given the prescribed conditions applied to the model along with the objective function and constraints. Your optimization must apply appropriate constraints and restrictions; otherwise, the Optimization module can extensively alter the topology of the component. The resolution of the structure that has been optimized with topology optimization is very dependent on the discretization. A fine mesh produces a structure with a higher resolution than a coarse mesh; however, it will also substantially increase the processing time required. You must determine the appropriate compromise between structural resolution and processing time.

During topology optimization the Optimization module modifies the material definition of the elements in the design area. As a result, you must provide the initial density of the materials in the design area, even if it is not required by the Abaqus analysis.

Restrictions on an Abaqus Model Used for Shape Optimization

Abaqus performs a shape optimization by modifying the boundaries or surfaces of a component. The optimization uses the stress condition to calculate new coordinates for nodes on the surface of the component and then adjusts the underlying mesh accordingly. The mesh quality must be sufficient to ensure that the analysis results are mostly unchanged by the movement of the surface nodes. High stress gradients must not be present within an element.

When the Optimization module is performing a shape optimization on a shell structure, it optimizes the form of the shell structure and not its thickness. The nodal position along shell edges can be modified; however, Abaqus does not modify the shell definition.

Restrictions on an Abaqus Model Used for Sizing Optimization

Abaqus performs a sizing optimization by modifying the thickness of shell elements in the design region. The element thickness must be uniform, and only single-layered shells are supported. Prescribed displacements are allowed in a static stress/displacement analysis; however, they are not supported in a frequency analysis.

Restrictions on an Abaqus Model Used for Bead Optimization

Abaqus performs a bead optimization by moving nodes of shell elements in the direction of the shell normal in the design region. The element thickness must be uniform, and only single-layered shells are supported. Prescribed displacements are allowed in a static stress/displacement analysis; however, they are not supported in a frequency analysis.

Supported Materials in the Design Area

The material models supported by structural optimization in the elements in the design area depend on the type of optimization—condition-based topology optimization, general topology optimization, or shape optimization.

Materials Supported by Condition-Based Topology Optimization

Condition-based topology optimization in Abaqus supports linear elastic, plastic, and hyperelastic material models.

Support for Linear Elastic Material Models

The following linear elastic material models are supported by condition-based topology optimization:

- Linear elastic materials with isotropic behavior.
- Linear elastic materials with fully anisotropic behavior.
- Linear elastic materials with orthotropic behavior. All of the behavior models are supported, except for orthotropic shear behavior for warping elements and coupled and uncoupled traction behavior for cohesive elements.

Support for Plastic Material Models

Metal plasticity material properties—the plastic part of the material model for elastic-plastic materials that use the Mises or Hill yield surface—are supported by condition-based topology optimization. Isotropic hardening is supported; however, cyclic loading is not supported—each material point can be unloaded only once and should not become elastoplastic again.

Support for Hyperelastic Material Models

All of the hyperelastic material models are supported by condition-based topology optimization, except for the Marlow material model and the hyperelastic material models with test data.

Support for Temperature and Field Variable Dependency

Condition-based topology optimization supports materials that have temperature and field variable dependency.

Materials Supported by General Topology Optimization

General topology optimization in Abaqus supports linear elastic, plastic, and hyperelastic material models.

Support for Linear Elastic Material Models

The following linear elastic material models are supported by general topology optimization:

- Linear elastic materials with isotropic behavior.
- Linear elastic materials with fully anisotropic behavior.
- Linear elastic materials with orthotropic behavior. All of the behavior models are supported, except for orthotropic shear behavior for warping elements and coupled and uncoupled traction behavior for cohesive elements.

Support for Plastic Material Models

Metal plasticity material properties—the plastic part of the material model for elastic-plastic materials that use the Mises or Hill yield surface—are supported by general topology optimization. Isotropic hardening is supported; however, cyclic loading is not supported—each material point can be unloaded only once and should not become elastoplastic again.

Support for Hyperelastic Material Models

All of the hyperelastic material models are supported by general topology optimization, except for the Marlow material model and the hyperelastic material models with test data.

Support for Temperature and Field Variable Dependency

Materials that have temperature and field variable dependency are supported by general topology optimization.

Material Support in Shape Optimization

All of the Abaqus material models are supported by shape optimization.

Material Support in Sizing Optimization

All of the Abaqus material models, including nonlinear materials, are supported inside and outside the design area. However, cyclic loading is not permitted.

Material Support in Bead Optimization

Nonlinear materials in the design area are not supported by bead optimization. All of the Abaqus material models, including nonlinear materials, are supported outside the design area.

Support for Coordinate Systems

In most cases, you will use the same coordinate system to define your model and the optimization task. However, the Optimization module allows you refer to a different coordinate system when you are defining a design response.

Supported Element Types

The Abaqus elements that are supported as design elements by topology and shape optimization are listed in *Table 1* through *Table 4*. The tables also list the Abaqus elements that support the reaction and internal force design responses. The shell elements that are supported as design elements by sizing and bead optimization are listed in *Table 5* and *Table 6*, respectively. Unsupported elements are ignored during optimization and remain unchanged. Structural optimization does not place any restrictions on the type of elements that you use outside the design area.

Supported Two-Dimensional Solid Elements

Topology optimization (both condition-based and general) and shape optimization support the two-dimensional solid elements listed in *Table 1*.

Table 1: Supported two-dimensional solid elements.

CPE3[1], CPE3H, CPE4[1], CPE4H, CPE4I, CPE4IH, CPE4R[1], CPE4RH,
CPE6H, CPE6M, CPE6MH
CPE8[1], CPE8H, CPE8R[1], CPE8RH
CPS3[1], CPS4[1], CPS4I, CPS4R[1], CPS6[1], CPS6M, CPS6MT, CPS8[1]. CPS8R[1]
CPEG3, CPEG3H, CPEG4, CPEG4H, CPEG4I, CPEG4IH, CPEG4R, CPEG4RH, CPEG6, CPEG6H, CPEG6M, CPEG6MH, CPEG8, CPEG8H, CPEG8R, CPEG8RH
CPE3T, CPE4T, CPE4HT, CPE4RT, CPE4RHT, CPE6MT, CPE6MHT, CPE8T, CPE8HT, CPE8RT, CPE8RHT
CPS3T, CPS4T, CPS4RT, CPS8T, CPS8RT
CPEG3T, CPEG3HT, CPEG4T, CPEG4RT, CPEG4RHT, CPEG6MT, CPEG6MHT, CPEG8T, CPEG8HT, CPEG8RHT
[1] Can include reaction and internal force design responses.

Supported Three-Dimensional Solid Elements

Topology optimization (both condition-based and general) and shape optimization support the three-dimensional solid elements listed in *Table 2*.

Table 2: Supported three-dimensional solid elements.

C3D4[1], C3D4H, C3D8[1]
C3D6[1], C3D6H
C3D8H, C3D8I, C3D8IH, C3D8R[1], C3D8RH
C3D10[1], C3D10H, C3D10M, C3D10MH
C3D15[1], C3D15H
C3D20[1], C3D20H, C3D20R[1], C3D20RH
C3D4T, C3D6T, C3D8T, C3D8HT, C3DHRT, C3D8RHT, C3D10MT, C3D10MHT, C3D20T, C3D20HT, C3D20RT, C3D20RHT
[1] Can include reaction and internal force design responses.

Supported Axisymmetric Solid Elements

Topology optimization (both condition-based and general) and shape optimization support the axisymmetric solid elements listed in *Table 3*.

Table 3: Supported axisymmetric solid elements.

CAX3[1], CAX3H, CAX4[1], CAX4H, CAX4I, CAX4IH, CAX4R[1], CAX4RH
CAX8[1], CAX8H, CAX8R[1], CAX8RH
CGAX3, CGAX3H, CGAX4, CGAX4H, CGAX4R, CGAX4RH, CGAX8, CGAX8H, CGAX8R, CGAX8RH
CAX3T, CAX4T, CAX4HT, CAX4RT, CAX4RHT, CAX8T, CAX8HT, CAX8RT, CAX8RHT
CGAX3T, CGAX3HT, CGAX4T, CGAX4HT, CGAX4RT, CGAX4RHT, CGAX8T, CGAX8HT, CGAX8RT, CGAX8RHT
[1] Can include reaction and internal force design responses.

Additional Supported Elements

Table 4 lists the general membrane, three-dimensional conventional shell, and beam elements that are supported by optimization.

Table 4: Additional supported elements

General membrane elements (topology and shape optimization)	M3D3[1], M3D4[1], M3D4R[1], M3D6[1], M3D8[1], M3D8R[1]
Three-dimensional conventional shell elements (topology optimization only)	STRI3, S3, S3R, STRI65, S4, S4R, S4R5, S8R, S8R5, S8RT
Three-dimensional conventional shell elements (shape optimization only)	STRI3[1], S3[1], S3R[1], S4[1], S4R[1], S8R[1]
Beam elements (shape optimization only)	B21[2], B21H[2], B31[2], B31H[2]
[1] Can include reaction and internal force design responses.	
[2] You can include beam elements in shape optimization only to define a neighboring component that is used to restrict the movement of nodes in the optimized region.	

Supported Three-Dimensional Conventional Shell Elements

Sizing optimization supports only the three-dimensional conventional shell elements listed in *Table 5*.

Table 5: Supported three-dimensional conventional shell elements for sizing optimization.

S3, S3R, S4, S4R, S8R
STRI65[1]
[1] You must request that rotational degrees of freedom be written to the output database.

Condition-based bead optimization supports all Abaqus plate and shell elements. However, general bead optimization supports only the three-dimensional conventional shell elements listed in *Table 6*.

Table 6: Supported three-dimensional conventional shell elements for general bead optimization.

S3, S3R
STRI3
S4, S4R
S8R

Eulerian Analysis Techniques

In this section:

- *Eulerian Analysis*
- *Defining Eulerian Boundaries*
- *Eulerian Mesh Motion*
- *Defining Adaptive Mesh Refinement in the Eulerian Domain*

Eulerian Analysis

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Eulerian Surface Definition*
- *Eulerian Elements*
- **EULERIAN SECTION*
- **INITIAL CONDITIONS*
- **SURFACE*
- *Creating Eulerian sections*
- *Defining a material assignment field*
- *Eulerian analyses*

Overview

In a traditional Lagrangian analysis nodes are fixed within the material, and elements deform as the material deforms. Lagrangian elements are always 100% full of a single material, so the material boundary coincides with an element boundary.

By contrast, in an Eulerian analysis nodes are fixed in space, and material flows through elements that do not deform. Eulerian elements may not always be 100% full of material—many may be partially or completely void. The Eulerian material boundary must, therefore, be computed during each time increment and generally does not correspond to an element boundary. The Eulerian mesh is typically a simple rectangular grid of elements constructed to extend well beyond the Eulerian material boundaries, giving the material space in which to move and deform. If any Eulerian material moves outside the Eulerian mesh, it is lost from the simulation.

Eulerian material can interact with Lagrangian elements through Eulerian-Lagrangian contact; simulations that include this type of contact are often referred to as coupled Eulerian-Lagrangian (CEL) analyses. This powerful, easy-to-use feature of Abaqus/Explicit general contact enables fully coupled multi-physics simulation such as fluid-structure interaction.

Applications

Eulerian analyses are effective for applications involving extreme deformation, up to and including fluid flow. In these applications, traditional Lagrangian elements become highly distorted and lose accuracy. Liquid sloshing, gas flow, and penetration problems can all be handled effectively using Eulerian analysis. Eulerian-Lagrangian contact allows the Eulerian materials to be combined with traditional nonlinear Lagrangian analyses.

An example of using Eulerian analysis for a severe deformation analysis is discussed in *Rivet forming*; using coupled Eulerian-Lagrangian contact for a fluid-structure interaction application is illustrated in *Impact of a water-filled bottle*.

Eulerian Volume Fraction

The Eulerian implementation in Abaqus/Explicit is based on the volume-of-fluid method. In this method, material is tracked as it flows through the mesh by computing its Eulerian volume fraction (EVF) within each element. By definition, if a material completely fills an element, its volume fraction is one; if no material is present in an element, its volume fraction is zero.

Eulerian elements may simultaneously contain more than one material. If the sum of all material volume fractions in an element is less than one, the remainder of the element is automatically filled with “void” material. Void material has neither mass nor strength.

Material Interfaces

Volume fraction data are computed for each Eulerian material in an element. Within each time increment, the boundaries of each Eulerian material are reconstructed using these data. The interface reconstruction algorithm approximates the material boundaries within an element as simple planar facets (the Eulerian method is implemented only for three-dimensional elements). This assumption produces a simple, approximate material surface that may be discontinuous between neighboring elements. Therefore, accurate determination of a material's location within an element is possible only for simple geometries, and fine grid resolution is required in most Eulerian analyses.

The discontinuities in an Eulerian material surface can lead to physically unrealistic configurations when visualizing the results of an Eulerian analysis. Abaqus/CAE can apply a nodal averaging algorithm to estimate a more realistic, continuous surface during visualization. For more information on visualizing the material interfaces in an Eulerian model, see *Viewing output from Eulerian analyses*.

Eulerian Section Definition

An Eulerian section definition lists all of the materials that may appear within an Eulerian element. Void material is automatically included in this list.

The material list supports an optional material instance name. Material instance names are required to uniquely identify materials that you use more than once. Repeated materials are useful, for example, in mixing simulations where the motion of a material interface is to be computed: the water in a tank could be divided by creating water material instances named “water_left” and “water_right,” and the evolution of the interface between these materials could be simulated.

By default, all Eulerian elements are initially filled with void material, regardless of the section assignment. You must introduce nonvoid material into your Eulerian mesh using an initial condition (see *Initial Conditions* below).

Eulerian Mesh Deformation

The Eulerian time incrementation algorithm is based on an operator split of the governing equations, resulting in a traditional Lagrangian phase followed by an Eulerian, or transport, phase. This formulation is known as “Lagrange-plus-remap.” During the Lagrangian phase of the time increment nodes are assumed to be temporarily fixed within the material, and elements deform with the material. During the Eulerian phase of the time increment deformation is suspended, elements with significant deformation are automatically remeshed, and the corresponding material flow between neighboring elements is computed.

At the end of the Lagrangian phase of each time increment, a tolerance is used to determine which elements are significantly deformed. This test improves performance by allowing those elements with little or no deformation to remain inactive during the Eulerian phase. The inactive elements typically have no impact on the visualization of an Eulerian analysis; however, plotting an Eulerian mesh using a very large deformation scale factor may reveal slight deformations for elements within the deformation tolerance.

Eulerian Material Advection

As material flows through an Eulerian mesh, state variables are transferred between elements by advection. The variables are assumed to be linear or constant in each old element, then these values are integrated over the new elements after remeshing. The new value of the variable is found by dividing the value of each integral by the material volume or mass in the new element.

Second-Order Advection

Second-order advection assumes a linear distribution of the variable in each old element. To construct the linear distribution, a quadratic interpolation is constructed from the constant values at the integration points of the middle element and its adjacent elements. A trial linear distribution is found by differentiating the quadratic function to find the slope at the integration point of the middle element. The trial linear distribution in the middle element is limited by reducing its slope until its minimum and maximum values are within the range of the original constant values in the adjacent elements. This process is referred to as flux limiting and is essential to ensure that the advection is monotonic.

Second-order advection is used by default, and it is recommended for all problems, ranging from quasi-static to transient dynamic shock.

Input File Usage: **EULERIAN SECTION*, ADVECTION=SECOND ORDER

Abaqus/CAE Usage: The second-order advection method is used by default in Abaqus/CAE.

First-Order Advection

First-order advection assumes a constant value of the variable in each old element. This method is simple and computationally efficient; however, it tends to diffuse sharp gradients over time. Therefore, this technique should be used only as a computationally efficient alternative for quasi-static simulations.

Input File Usage: **EULERIAN SECTION*, ADVECTION=FIRST ORDER

Abaqus/CAE Usage: The first-order advection method cannot be specified in Abaqus/CAE.

Reducing the Stable Time Increment Based on the Advection Speed

The stable time increment size is adjusted automatically to prevent material from flowing across more than one element in each increment. When the material velocity approaches the speed of sound (for example, in simulations involving blast and shocks), further restrictions on the time increment size may be needed to maintain accuracy and stability. You can specify a flux limit ratio to restrict the stable time increment size such that material can flow across only a fraction of an element in each increment. The default flux limit ratio is 1.0, and recommended values range from 0.1 to 1.0.

Input File Usage: **EULERIAN SECTION*, FLUX LIMIT RATIO=*maximum ratio*

Abaqus/CAE Usage: The flux limit ratio cannot be modified in Abaqus/CAE.

Initial Conditions

You can apply initial conditions to Eulerian nodes and elements in the same way that they are used for Lagrangian nodes and elements. Initial stress, temperature, and velocity are common examples. In addition, most Eulerian analyses require the initialization of Eulerian material.

By default, all Eulerian elements are initially void. You can use initial conditions to fill Eulerian elements with one or more of the materials listed in the Eulerian section definition. By selectively filling elements, you can create the initial shape of each Eulerian material.

To fill an Eulerian element, you must define an initial volume fraction for each available material instance. Material is filled until a volume fraction of 1.0 is reached; any excess material is ignored. The initial conditions

apply only at the beginning of an analysis; during the analysis the materials deform according to the applied loads, and the volume fractions are recalculated accordingly.

Input File Usage: **INITIAL CONDITIONS*, TYPE=VOLUME FRACTION

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Material Assignment** for the **Types for Selected Step**

Boundary Conditions

By default, Eulerian material can flow freely into and out of the Eulerian domain through mesh boundaries. You can constrain degrees of freedom at Eulerian nodes to restrict material flow. For example, you can define typical fluid "stick" or "sliding" walls using constraints normal and/or tangential to the boundary. Since Eulerian nodes are automatically repositioned during the Eulerian transport phase, you cannot apply prescribed displacement boundary conditions to them.

You can use prescribed velocity or acceleration conditions on Eulerian nodes to control material flow. Prescribed velocity or acceleration is implemented in an Eulerian frame, so material velocity will reach the prescribed value as the material passes the Eulerian node. If velocity is directed outward at an Eulerian mesh boundary, either by prescribed condition or naturally as a result of dynamic equilibrium, material may flow out of the Eulerian domain. This material is lost from the simulation, and corresponding decreases in total mass and energy will occur.

Similarly, if velocity is directed inward at a boundary, inflow of material into the Eulerian domain will occur. When materials flow into an element through a boundary face, the material content and the state of each inflowing material are equal to that which presently exists within the element. For example, if a boundary element contains 60% hot water and 40% cold air and the interface normal is parallel to the boundary face, inflow velocity will introduce a mixture of 60% hot water and 40% cold air. In this case corresponding increases in total mass and energy will occur.

You can also define inflow and outflow conditions at an Eulerian domain boundary, as described in *Defining Eulerian Boundaries*.

Loads

You can apply loads to Eulerian nodes, elements, and faces in the same way as to their Lagrangian counterparts. Eulerian loads act in an Eulerian frame: they affect Eulerian material as it passes the point of load application.

You can also apply surface tension loads to an Eulerian material surface that involves a specific Eulerian material instance (see *Eulerian Surface Definition*).

Material Options

You can define material properties for Eulerian analysis in the same way as for Lagrangian analysis. Liquids and gases can be modeled using equation of state materials (see *Equation of State*). Brittle cracking is not supported because the number of cracks is not a continuous quantity and cannot be easily remapped. Hyperelastic materials can be used in an Eulerian analysis, but due to inaccuracies introduced to the deformation gradient during material transport, these materials might not fully recover their original configuration after loads are removed; the same inaccuracies also affect user-defined materials.

Eulerian analysis allows materials to undergo extreme strain without the mesh distortion limitations of Lagrangian analysis. Therefore, it is especially important to define your material behavior through the entire strain range, which often requires definition of a failure behavior.

Isotropic material failure is supported using a damage variable to characterize the failure level. Element deletion is suppressed for Eulerian sections because undamaged material may flow into “failed” elements. Shear failure models are not supported.

Rayleigh mass proportional damping is not supported.

Elements

The Eulerian method is implemented in the multi-material element type EC3D8R and the multi-material thermally coupled element type EC3D8RT. The underlying mechanical response formulation of these elements is based on the Lagrangian C3D8R element with extensions to allow multiple materials and to support the Eulerian transport phase. The formulation applies the same strain to each material in the element, then allows the stress and other state data to evolve independently within each material. These stresses are combined using volume fraction data to create element averaged values, which are integrated to obtain nodal forces. Similarly, the thermal response formulation for the thermally coupled element is based on the Lagrangian element C3D8RT with the extension to allow multiple materials with different thermal properties and to support temperature advection. All the materials have the same temperature, and the thermal properties (such as thermal conductivity and thermal capacitance) are volume averaged before being used in solving one single heat transfer equation for the multi-material model.

Element averaged values of other state data are computed similarly for output purposes.

The Eulerian EC3D8R and EC3D8RT elements require eight nodes. Degenerate elements are not supported. The Eulerian method is not implemented for two-dimensional elements. Axisymmetry can be simulated using a wedge-shaped mesh and symmetry boundary conditions.

By default, the Eulerian elements use viscous hourglass control. Hourglass control is disabled by default for incompressible liquids modeled using equation of state material types. These choices can be modified using section controls (see *Section Controls*).

Constraints

Since Eulerian nodes are automatically repositioned during the Eulerian transport phase, you cannot use Eulerian nodes in Lagrangian modeling features such as elements, connectors, and constraints. However, bonding at an Eulerian-Lagrangian contact interface can be modeled with general contact by assigning no separation and rough friction contact behavior (see *Interactions*, *Defining General Contact in Abaqus/Explicit*, *Using the No Separation Relationship*, and *Preventing Slipping regardless of Contact Pressure*).

Interactions

Eulerian material instances interact with each other with a sticky behavior. This sticking occurs because of the kinematic assumption that a single strain field is applied to all materials within an element. Tensile stress can be transmitted across an interface between two Eulerian materials, and no slip occurs at these interfaces. This Eulerian-to-Eulerian contact behavior can be reasonable in some situations, such as in a simulation of a lead bullet penetrating a steel plate. Ablation of the bullet surface against the steel is captured by the sticky behavior within the Eulerian elements at the bullet-steel interface. Relative motion along this interface will occur only due to shearing of the lead material.

Eulerian-to-Eulerian contact occurs by default in an Eulerian analysis; you do not need to define contact interactions between Eulerian materials.

More complex contact interactions can be simulated when one of the contacting bodies is modeled using Lagrangian elements. This powerful capability supports applications such as fluid-structure interaction, where an Eulerian fluid contacts a Lagrangian structure.

The implementation of Eulerian-Lagrangian contact is an extension of general contact in Abaqus/Explicit. The general contact property models and defaults apply to Eulerian-Lagrangian contact (see *About Mechanical Contact Properties*). For example, by default, tensile stresses are not transmitted across an Eulerian-Lagrangian contact interface, and the interface friction coefficient is zero. Specifying automatic contact for an entire Eulerian-Lagrangian model allows for interactions between all Lagrangian structures and all Eulerian materials in the model. You can also use Eulerian surfaces (see *Eulerian Surface Definition*) to create material-specific interactions or to exclude contact between particular Lagrangian surfaces and Eulerian materials.

Input File Usage: Use both of the following options to define contact between all Lagrangian bodies and all Eulerian materials:

**CONTACT*
**CONTACT INCLUSIONS*, ALL EXTERIOR

Use the following options to include or exclude contact between particular Lagrangian surfaces and Eulerian materials:

**CONTACT*
**CONTACT INCLUSIONS*
Lagrangian_surface, *Eulerian_surface*
**CONTACT EXCLUSIONS*
Lagrangian_surface, *Eulerian_surface*

Abaqus/CAE Usage: Use the following option to define contact between all Lagrangian bodies and all Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**: **Included surface pairs: All* with self**

Use the following options to include contact between particular Lagrangian surfaces and Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**:
Included surface pairs: Selected surface pairs: Edit, select the Lagrangian surface in the left column and the Eulerian material instance in the right column, then click the arrows to transfer them to the list of included pairs

Use the following options to exclude contact between particular Lagrangian surfaces and Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**:
Excluded surface pairs: Edit, select the Lagrangian surface in the left column and the Eulerian material instance in the right column, then click the arrows to transfer them to the list of excluded pairs

Formulation of Eulerian-Lagrangian Contact

The Eulerian-Lagrangian contact formulation is based on an enhanced immersed boundary method. In this method the Lagrangian structure occupies void regions inside the Eulerian mesh. The contact algorithm automatically computes and tracks the interface between the Lagrangian structure and the Eulerian materials. A great benefit of this method is that there is no need to generate a conforming mesh for the Eulerian domain. In fact, a simple regular grid of Eulerian elements often yields the best accuracy.

If the Lagrangian body is initially positioned inside the Eulerian mesh, you must make sure that the underlying Eulerian elements contain void after material initialization. During the analysis the Lagrangian body pushes

material out of the Eulerian elements that it passes through, and they become filled with void. Similarly, Eulerian material flowing toward the Lagrangian body is prevented from entering the underlying Eulerian elements. This formulation ensures that two materials never occupy the same physical space.

If the Lagrangian body is initially positioned outside the Eulerian mesh, at least one layer of void Eulerian elements must be present at the Eulerian mesh boundary. This creates a free surface on the Eulerian material inside the Eulerian mesh boundary and provides a source for void material to replace Eulerian material that is driven out of interior elements. Several layers of void elements are typically used above free surfaces to allow simulation of crater formation and backsplashing before this material leaves the Eulerian domain.

Eulerian-Lagrangian contact also supports failure and erosion in the Lagrangian body. Lagrangian element failure can open holes in a surface through which Eulerian material may flow. When modeling erosion of a solid Lagrangian body, the interior faces of the solid body must be included in the contact surface definition (see *Modeling Surface Erosion*).

Eulerian-Lagrangian contact constraints are enforced using a penalty method, where the default penalty stiffness parameter is automatically maximized subject to stability limits.

The default contact formulation is effective for modeling solid materials but may exhibit leakage when modeling liquids or gases. An enhanced Eulerian-Lagrangian contact formulation is available to prevent the leakage of liquids or gases through the Lagrangian surface. It is more computationally expensive and should be activated only when required.

Eulerian-Lagrangian contact supports thermal interactions when using coupled temperature-displacement Eulerian element EC3D8RT in a dynamic coupled thermal-stress analysis. However, gap radiation and gap conductance as a function of clearance are not supported.

Output

The set of element output variables EVF gives the Eulerian volume fraction for each material in the Eulerian section definition, including void. It is important to request output for EVF in all Eulerian analyses because visualization of Eulerian material boundaries is based on the material volume fractions.

Material-specific Eulerian field output variables are distinguished by appending material names to the base field name. For example, if you request output variable S (stress components) in an Eulerian analysis involving material instances named “steel” and “tin,” you will see results for individual material stresses named “S_steel” and “S_tin.”

Several volume fraction averaged field data are also available for output. For example, output variable SVAVG gives a single value of stress for each element computed as a volume fraction average of stress over all materials present in the element. Use of these volume fraction averaged output data has the advantage of substantially reducing the size of the output database for the case where several materials are defined in the Eulerian section. See *Abaqus/Explicit Output Variable Identifiers* for a complete list of Eulerian-specific output variables.

Output variables EVF and SVAVG are included in the PRESELECT variable list when Eulerian elements appear in the model.

You can also request integrated volume (VOLEUL) and integrated mass (MASSEUL) over a particular Eulerian element set. These output variables are material specific and are distinguished by having the material names appended to the variable name.

Limitations

Eulerian analyses are subject to the following limitations:

- Boundary conditions: You cannot apply prescribed nonzero displacement boundary conditions to Eulerian nodes.

- Lagrangian attachments: You cannot attach Lagrangian elements to Eulerian nodes. Use tied contact interfaces instead.
- Constraints: You cannot apply Lagrangian constraints (MPCs, etc.) to Eulerian nodes. Use tied contact interfaces instead.
- Materials: Brittle cracking and shear failure models are also not supported. Rayleigh mass proportional damping is not supported.
- Elements: The Eulerian formulation is implemented only for EC3D8R and EC3D8RT elements.
- Element import: Eulerian elements are not available for import.
- Double-sided contact: Penetration of Eulerian material through the contact interface can occur in some cases involving Eulerian material contacting Lagrangian shell or membrane elements. This type of contact introduces complexity because the sign of the outward normal direction must be determined on the fly as material approaches the Lagrangian element; contact with either side of the element is potentially allowable. You should simplify the contact problem wherever possible by using Lagrangian solid elements instead of shell or membrane elements, since the outward normal direction at solid element faces is unique. For example, if a model involves Eulerian material flowing around a rigid Lagrangian obstacle, mesh the obstacle with solid elements rather than shell elements and activate the enhanced contact formulation to reduce the leakage.
- Contact penetration: In some cases Eulerian material may penetrate through the Lagrangian contact surface near corners. This penetration should be limited to an area equal to the local Eulerian element size. Penetration can be minimized by refining the Eulerian mesh (manually or through adaptive mesh refinement) or by adding a fillet to the Lagrangian mesh with radius equal to the local Eulerian element size. Alternatively, you can activate the enhanced contact formulation to reduce or eliminate the leakage.
- When the enhanced contact formulation through adaptive mesh refinement is activated, you should simplify the Lagrangian contact surface wherever possible by avoiding double-sided, folded, and T-section surfaces. If more than one surface cuts through a single Eulerian element, the enhanced contact formulation is turned off, and the default formulation is used. For thin structures modeled with solid elements, refining the Eulerian element size to avoid this behavior is recommended.
- Contact types: Eulerian-Lagrangian contact does not support Lagrangian beam elements, Lagrangian pipe elements, Lagrangian truss elements, or analytical rigid surfaces.
- Contact import: Import of the Eulerian-Lagrangian contact states is not supported.
- Thermal contact: Gap radiation and gap conductance as a function of clearance are not supported.
- Contact output: Contact variables are output only for the Lagrangian side of Eulerian-Lagrangian interfaces.
- Surface loads: You cannot use the Eulerian material surface type for general surface loading. However, distributed loads such as pressure can be applied to surfaces defined on Eulerian element faces.
- Surface tension loads: You cannot apply surface tension loads to an Eulerian material surface when adaptive mesh refinement is activated.
- Mass scaling: You cannot apply mass scaling to Eulerian elements.
- Heat transfer: Use coupled temperature-displacement EC3D8RT Eulerian elements to model a fully coupled thermal-stress analysis. Adiabatic conditions are assumed in Eulerian materials when EC3D8R elements are used.
- Output: Total strain (LE) is not available for Eulerian elements in field or history output, but it can be accessed via the utility routine `VGETVRM`.
- Subcycling: You cannot include Eulerian elements in subcycling zones.

Input File Template

The following example illustrates a coupled Eulerian-Lagrangian analysis of a Lagrangian boat floating on Eulerian water. A conforming mesh is assumed, so Eulerian material initialization is achieved by whole element filling. Material-specific interactions between the Lagrangian body and the Eulerian materials are implemented: a contact interaction is defined between the boat and water, but contact between the boat and air is ignored. Output is requested for Eulerian volume fractions, Eulerian element-averaged stress, and material stress.

```
*HEADING
…
*ELEMENT, TYPE=C3D8R, ELSET=BOAT_ELSET
element definitions for Lagrangian boat
*ELEMENT, TYPE=EC3D8R, ELSET=ALL_EULERIAN
element definitions for whole Eulerian mesh
*ELSET, NAME=AIR_ELSET
data lines giving Eulerian elements that are initially filled with air
*ELSET, NAME=WATER_ELSET
data lines giving Eulerian elements that are initially filled with water
**
*MATERIAL, NAME=AIR
material definition for air
*MATERIAL, NAME=WATER
material definition for water
**
*EULERIAN SECTION, ELSET=ALL_EULERIAN
AIR
WATER
**
*INITIAL CONDITIONS, TYPE=VOLUME FRACTION
AIR_ELSET, AIR, 1.0
WATER_ELSET, WATER, 1.0
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC
data lines to define water pressure due to gravity
**
*SURFACE, NAME=WATER_SURFACE, TYPE=EULERIAN MATERIAL
WATER
*SURFACE, NAME=BOAT_SURFACE
BOAT_ELSET
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
BOAT_SURFACE, WATER_SURFACE
**
*OUTPUT, FIELD
*ELEMENT OUTPUT
EVF, SVAVG, PEEQVAVG
*END STEP
```

References

- Benson, D. J., “Computational Methods in Lagrangian and Eulerian Hydrocodes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 99, pp. 235–394, 1992.

- Benson, D. J., "Contact in a Multi-Material Eulerian Finite Element Formulation," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, pp. 4277–4298, 2004.
- Peery, J. S., and D. E. Carroll, "Multi-Material ALE methods in Unstructured Grids," *Computer Methods in Applied Mechanics and Engineering*, vol. 187, pp. 591–619, 2000.

Defining Eulerian Boundaries

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Eulerian Analysis*
- *Eulerian Elements*
- **EULERIAN BOUNDARY*
- *Defining an Eulerian boundary condition*

Overview

In an Eulerian analysis you can define independent inflow and outflow conditions at an Eulerian boundary. An Eulerian boundary condition:

- can be used to control the flow of material into the Eulerian domain;
- can be used to define a pressure field at the boundary of an Eulerian domain;
- can be used to apply a nonreflecting boundary condition at the truncated artificial boundary to simulate an infinite domain; and
- is associated with a surface defined on the Eulerian mesh boundary where inflow or outflow occurs.

Defining the Eulerian Boundary

Eulerian boundaries must be defined at surfaces on the Eulerian mesh boundary. You cannot define multiple Eulerian boundaries at the same surface.

Input File Usage: **EULERIAN BOUNDARY*
surface name

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: **Category: Other**, **Types for Selected Step: Eulerian boundary**: select region

Defining the Inflow Condition

You can use the inflow condition to control the flow of material into the Eulerian domain.

Free Inflow

If no Eulerian boundary is defined, material can flow into the Eulerian domain freely; and the material content and the state of each inflow material are equal to that which presently exists within the element. If an Eulerian boundary is defined, free inflow is the default inflow condition.

Input File Usage: **EULERIAN BOUNDARY*, INFLOW=FREE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: Free**

No Inflow

You can specify an Eulerian boundary where no inflow can occur—no material or void can flow into the Eulerian domain through the specified boundary. The normal component of the velocity is set to zero if the velocity is directed inward at the boundary, while the tangential component of the velocity remains unchanged.

Input File Usage: **EULERIAN BOUNDARY*, INFLOW=NONE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: None**

Void Inflow

You can also specify a boundary through which inflow can occur but the influx volume contains only void. Due to the inflow of void, an Eulerian domain that is initially completely full of material might become partially full during the analysis.

Input File Usage: **EULERIAN BOUNDARY*, INFLOW=VOID

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: Void**

Defining the Outflow Condition

The outflow condition can be used to simulate an unbounded domain by reducing reflection at the outflow boundary or to prescribe a pressure field at the boundary.

Free Outflow

If no Eulerian boundary condition is specified, material can flow out of the Eulerian domain freely; and the material content and the state of each outflow material are equal to that which presently exists within the element. If an Eulerian boundary condition is defined, free outflow is the default behavior if the void inflow condition is specified at the same surface.

Input File Usage: **EULERIAN BOUNDARY*, OUTFLOW=FREE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Free**

Nonreflecting Outflow

A nonreflecting outflow condition can be used in boundary value problems defined in unbounded domains or problems in which the region of interest is small in size compared to the surrounding medium. Like the infinite element formulation described in *Using Solid Medium Infinite Elements in Dynamic Analyses*, the nonreflecting outflow condition introduces additional normal and shear tractions on the domain boundary that are proportional to the normal and shear components of the velocity of the boundary. These boundary damping constants are chosen to minimize the reflection of dilatational and shear wave energy back into the finite element mesh. This condition does not provide perfect transmission of energy out of the mesh except in the case of plane body waves impinging orthogonally on the boundary in an isotropic medium. However, it usually provides acceptable modeling for most practical cases. An exception is the case when significant material transport occurs through the boundary, in which case this condition is not suitable to be used.

Input File Usage: **EULERIAN BOUNDARY*, OUTFLOW=NONREFLECTING

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Nonreflecting**

Equilibrium Outflow

Equilibrium outflow is another outflow condition that can effectively reduce spurious reflection at artificial outflow boundaries in unbounded domains. It is assumed that the stress is zero-order continuous across the element faces on the boundary. Traction is applied to these element faces to balance the nodal forces created by the stress in the boundary elements. This condition is usually applied at the outflow boundary where the pressure distribution is unknown.

Input File Usage: **EULERIAN BOUNDARY*, OUTFLOW=NONUNIFORM PRESSURE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Equilibrium**

Zero-Pressure Outflow

It is common in flow problems to specify a zero pressure at the outlet of the flow. Since the normal traction on the boundary contains the contribution from both the pressure and the shear stress, the natural boundary condition, also known as the "do-nothing condition," is not sufficient to provide such a condition if the shear behavior of the flow is also considered. The zero pressure outflow condition applies a traction that counteracts the shear contribution and, thus, generates a uniformly distributed pressure field on the boundary. You can apply a distributed surface load (see *Surface Tractions and Pressure Loads*) on the same boundary to specify a nonzero pressure. This is the default outflow condition if the inflow condition is not specified.

Input File Usage: **EULERIAN BOUNDARY*, OUTFLOW=ZERO PRESSURE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Zero pressure**

Applying Fixed Boundary Conditions in the Normal Direction

You can specify an Eulerian boundary where the flow direction is tangential to the specified boundary surface. Eliminate the normal component of the flow by combining the no inflow and no outflow conditions.

Input File Usage: **EULERIAN BOUNDARY*, INFLOW=NONE,OUTFLOW=NONE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: None, Outflow**, **Outflow: None**

Using Eulerian Boundaries in Restart Analyses

You can define a new Eulerian boundary in a restart analysis, but you cannot specify a void inflow condition at this boundary. In addition, you cannot change the inflow condition at an existing Eulerian boundary to the void inflow condition in a restart analysis.

Eulerian Mesh Motion

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Eulerian Surface Definition*
- *Eulerian Analysis*
- **EULERIAN MESH MOTION*
- **EULERIAN SECTION*
- **SURFACE*
- *Defining an Eulerian mesh motion boundary condition*

Overview

In a traditional Eulerian analysis, material flows through an Eulerian mesh that is fixed in space. Since it is stationary, the Eulerian mesh must be large enough to enclose the entire trajectory of interest. In some simulations, such as a tumbling liquid-filled bottle, this trajectory can be long, requiring a large Eulerian mesh whose elements are mostly empty. The Eulerian mesh motion feature allows the Eulerian mesh to move in space, following, expanding, and contracting to enclose a target object. This can greatly reduce mesh size and, hence, simulation cost. Mesh motion can also simplify modeling by ensuring that the entire trajectory of interest, which may be unpredictable, is indeed covered by the Eulerian mesh.

Activating Mesh Motion

You can independently activate mesh motion for each Eulerian section in a model. The motion applies to all of the elements in the section.

Input File Usage: **EULERIAN MESH MOTION*, ELSET=*name*

Abaqus/CAE Usage: Load module: **BC->Create**, **Category: Other**, **Types for Selected Step: Eulerian mesh motion**: select an Eulerian part instance

Computing Mesh Motion

The motion of the Eulerian mesh is computed using an internally constructed bounding box that encloses the entire Eulerian section. The bounding box has six degrees of freedom: translation of the box center and scaling of each of the three box dimensions.

The bounding box is constructed in a local coordinate system. Its six degrees of freedom are also defined in this local system. The local coordinate directions remain fixed in space during the simulation. If no local coordinate system is specified, the local system coincides with the global system.

Input File Usage: **EULERIAN MESH MOTION*, ORIENTATION= *name*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Bounding Box Csys**: **Edit** or **Create**

Defining the Target Object

You use a surface to define the target object that the Eulerian mesh will follow. By default, the Eulerian mesh bounding box (and, hence, the Eulerian mesh) moves to enclose the surface at all times, subject to any constraints specified on the mesh motion. If the surface type is Lagrangian, the Eulerian mesh bounding box moves to enclose the surface nodes (see *Figure 1*). If the surface type is Eulerian, the Eulerian mesh bounding box moves to enclose the Eulerian material named in the surface definition (see *Figure 2*).

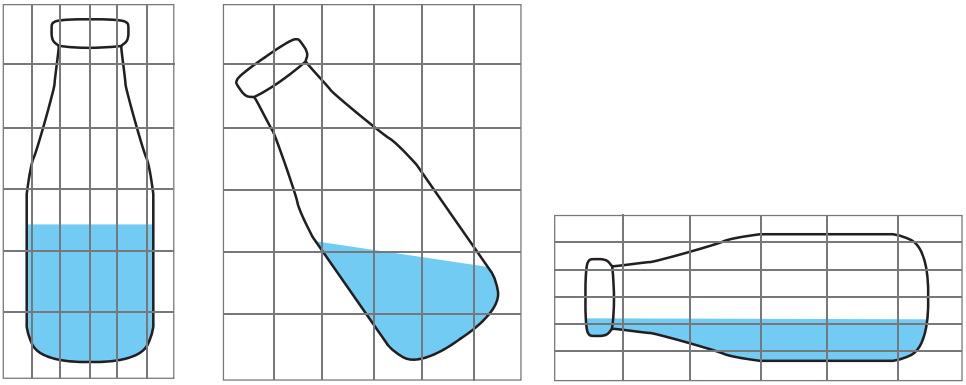


Figure 1: Mesh motion, where the target object is the Lagrangian bottle.

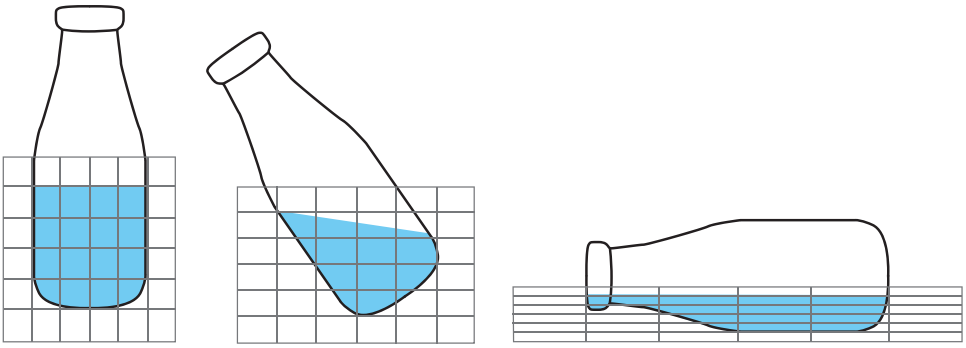


Figure 2: Mesh motion, where the target object is the Eulerian liquid.

The Eulerian mesh may not fully enclose the target object due to:

- constraints on the bounding box motion;
- a misalignment of the bounding box local orientation;
- a mismatch between the shape of the mesh boundary and the bounding box (i.e., the Eulerian mesh is not a rectangular box); or
- an inadequately sized or positioned initial Eulerian mesh.

Input File Usage: **EULERIAN MESH MOTION*, SURFACE=*name*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Object to Follow**: *name*

Constraining Eulerian Mesh Motion

Once the motion of the bounding box is computed, the translations and scaling factors are applied directly to the Eulerian mesh. Several types of constraints are available to restrict these motions. Conflicts between competing constraints are resolved in the following order of precedence:

1. constraining the center and faces of the mesh bounding box,
2. limiting the rate of mesh motion,
3. turning off mesh contraction,
4. centering the mesh bounding box on the target's center of mass or bounding box center,
5. preventing mesh expansion or contraction outside the scale factor limits,
6. limiting aspect ratio changes, and
7. maintaining a buffer between the mesh and target.

Constraining Mesh Expansion and Contraction

By default, the Eulerian mesh may expand or contract by an unlimited amount in each direction, as necessary to contain the target object. This can be undesirable: expansion creates large Eulerian elements that crudely approximate the shape of Eulerian objects, while contraction leads to decreased stable time increment sizes.

You can apply constraints to limit the expansion and contraction independently in each local direction by specifying lower and/or upper limits on the bounding box size scale factors. For example, a maximum scale factor of 1.0 constrains the box dimension to be no larger than 1.0 times the initial box dimension, effectively prohibiting any expansion, while a minimum scale factor of 0.5 limits the box dimension to be no smaller than half its initial dimension.

Input File Usage: **EULERIAN MESH MOTION*
scaling factor limits

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Axis *n***:
Expansion ratio, **Contraction ratio**

Preventing Mesh Contraction

An additional control is available to prevent incremental contraction. If specified, the box dimensions may increase, but at no point during the simulation may they decrease below their current values. This option prevents oscillations in mesh size during simulations where the mesh is nominally expanding.

Input File Usage: **EULERIAN MESH MOTION*, CONTRACT=NO

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**:
toggle off **Allow mesh contraction**

Constraining Mesh Translation

You can specify the motion of the center of the bounding box to be either free (default) or fixed in each of the local directions. You can also independently specify free (default) or fixed normal motion of the positive and negative box faces in the local coordinate directions.

Input File Usage: **EULERIAN MESH MOTION*
,
face constraints
center constraints

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Axis *n***: **Center position**, **Positive plane position**, **Negative plane position**

Centering the Mesh Bounding Box

If the motion of the mesh bounding box is unconstrained, the center of the bounding box is aligned with the center of a box enclosing the target surface. If the target surface fragments or "emits" low density material, aligning the center of the bounding box with the center of mass of the target may be advantageous.

Input File Usage: Use the following option to center the mesh bounding box on the center of mass of the target object:

**EULERIAN MESH MOTION*, CENTER=MASS

Use the following option to center the mesh bounding box on the center of the target object's bounding box:

**EULERIAN MESH MOTION*, CENTER=BOUNDING BOX

Abaqus/CAE Usage: The center of the mesh bounding box cannot be changed in Abaqus/CAE; the center of the mesh bounding box corresponds to the center of the target object's bounding box.

Controlling the Mesh Buffer around the Target Object

The mesh moves to maintain a buffer of Eulerian elements between the target object and the bounding box. By default, this buffer is equal to twice the maximum Eulerian element size in the mesh. You can specify the buffer size as a multiple of the maximum Eulerian element size. You can also specify that the initial spacing between the target object and the mesh (set to zero where the target initially extends outside of the mesh) is used to compute the buffer size.

Input File Usage: Use the following option to use a buffer equal to the initial spacing between the target object and the mesh:

**EULERIAN MESH MOTION*, BUFFER=INITIAL

Use the following option to specify a buffer as a multiple of the maximum Eulerian element size:

**EULERIAN MESH MOTION*, BUFFER= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Buffer size: Initial** or **Specify**

Limiting Aspect Ratio Changes

Excessive mesh motion in a single direction can produce badly shaped Eulerian elements. An optional parameter is available to limit the change in maximum aspect ratio of the bounding box. By default, this limit is 10. When

the aspect ratio limit is reached, motion in one local direction will induce motion in the other directions to preserve the box aspect ratio. This aspect ratio limit applies to the bounding box dimensions, not the underlying Eulerian element dimensions.

Input File Usage: **EULERIAN MESH MOTION*, ASPECT RATIO MAX= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Aspect ratio limit:** *value*

Limiting the Rate of Mesh Motion

The Eulerian mesh must not be allowed to move abruptly. A hard limit on its motion is given by the advective Courant condition, which prohibits mesh velocity larger than the material wave speed. In addition you can limit the mesh velocity to a multiple of the maximum velocity in the target object. By default, this limit is set to 1.01.

Input File Usage: **EULERIAN MESH MOTION*, VMAX FACTOR= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Mesh velocity factor:** *value*

Ignoring Fragments of Eulerian Material

When the target object is an Eulerian material, tiny fragments can drive excessive mesh motion. You can specify a minimum Eulerian volume fraction below which Eulerian material is ignored during the mesh motion calculation. This can be particularly useful for impact calculations, where tiny fragments of an impacting, splattering projectile may be allowed to leave the Eulerian domain. The default minimum volume fraction is 0.5.

Input File Usage: **EULERIAN MESH MOTION*, VOLFRAC MIN= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Volume fraction threshold:** *value*

Limitations

An Eulerian mesh can move only according to the available Eulerian mesh motion options. You cannot apply prescribed displacement boundary conditions to Eulerian nodes.

Defining Adaptive Mesh Refinement in the Eulerian Domain

Products: Abaqus/Explicit

References:

- *Eulerian Analysis*
- **ADAPTIVE MESH REFINEMENT*
- **EULERIAN SECTION*
- **EULERIAN MESH MOTION*
- **CONTACT CONTROLS ASSIGNMENT*

Overview

The adaptive mesh refinement feature:

- can refine elements locally inside an Eulerian mesh;
- allows you to define various criteria for refinement;
- can remove the refinement automatically once the refinement criteria are no longer met;
- allows you to activate the enhanced contact formulation for Eulerian-Lagrangian contact interfaces; and
- is available for Eulerian element type EC3D8R only.

Adaptive Mesh Refinement

In a traditional Eulerian analysis the topology of the Eulerian mesh does not change during the analysis. Although the Eulerian mesh motion feature allows the Eulerian mesh to move in space to cover areas of interest, its ability to create a nonuniformly refined mesh that changes with time is limited. The adaptive mesh refinement feature can locally refine the mesh by subdividing elements identified by user-defined criteria. This refinement can be removed automatically during the analysis once the criteria are no longer satisfied. This feature offers great savings in computational cost compared to a uniformly refined mesh. See *Impact of a copper rod* for an example of using the adaptive mesh refinement feature.

In some cases Eulerian material may penetrate through the Lagrangian contact surface, which could result in the leakage of liquid or gas through the Lagrangian contact surface. To prevent this, an enhanced contact formulation is available to improve the contact between the Eulerian material and the Lagrangian contact surface. This enhanced contact formulation is more expensive than the default contact algorithm and should be activated only when required.

Activating Adaptive Mesh Refinement

You can independently activate adaptive mesh refinement for each Eulerian section in a model. The feature applies to all the elements specified in the element set; all the elements in the element set have to be in the same Eulerian section.

Input File Usage: **ADAPTIVE MESH REFINEMENT*, ELSET=*name*

Activating the Enhanced Contact Formulation

You can activate the enhanced contact formulation for an Eulerian domain. However, if you use the enhanced contact formulation, you cannot specify adaptive mesh refinement criteria.

Input File Usage: Use the following option to activate the enhanced contact formulation:

**ADAPTIVE MESH REFINEMENT*, ENHANCED CONTACT=YES

Use the following option to deactivate the enhanced contact formulation:

**ADAPTIVE MESH REFINEMENT*, ENHANCED CONTACT=NO (default)

Setting the Refinement Limit

When adaptive mesh refinement occurs, elements are added to the Eulerian mesh. You can limit how many elements can be created by specifying an upper bound ratio of added elements to original elements. The default value of this upper bound ratio is 8.0 if you deactivate the enhanced contact formulation, and the default value is 1.0 if you activate the enhanced contact formulation. When you activate the enhanced contact formulation, Abaqus/Explicit adds elements along the Eulerian-Lagrangian interface only if they are needed. You can reduce the computational cost of the analysis by limiting how many elements are created.

Input File Usage: **ADAPTIVE MESH REFINEMENT*, RATIO=*maximum increase in number of elements/original number of elements*

Setting the Refinement Level

With one level of refinement, each time a user-defined Eulerian element is refined, it is equally divided into eight subelements. These subelements can subsequently be divided again if you allow two levels of refinement and so forth. You can set a limit on the maximum number of levels of refinement. The default maximum level is one. This setting is relevant only when you deactivate the enhanced contact formulation.

Input File Usage: **ADAPTIVE MESH REFINEMENT*, LEVEL=*maximum level of refinement*

Deactivating Coarsening

You can specify whether refinement can be removed when the refinement criteria are no longer met. This setting is relevant only when the enhanced contact formulation is deactivated.

Input File Usage: Use the following option to specify that refinement can be removed once the refinement criteria are no longer met:
**ADAPTIVE MESH REFINEMENT*, COARSENING=YES (default)
Use the following option to specify that refinement cannot be removed even when the refinement criteria are no longer met:
**ADAPTIVE MESH REFINEMENT*, COARSENING=NO

Defining Refinement Criteria

You must specify at least one refinement criterion unless the enhanced contact formulation is activated. An element will be selected for refinement if any of the criteria is met. To reduce the numerical artifacts at the mesh transition boundaries (where a fine mesh meets a coarse mesh), the elements adjacent to the selected elements are also refined. The elements are coarsened once the refinement criteria are no longer met. Each selected element can be refined or coarsened by only one level in every increment. *Table 1* lists all the refinement criteria available in Abaqus/Explicit.

Table 1: Refinement criteria.

Refinement criterion description	Refinement criterion label	User-specified values
Refine elements containing material interfaces	VF	N/A
Refine elements that are in contact with Lagrangian bodies	CONT	You can specify the value ALL to refine all elements intersecting the Lagrangian surfaces even if contact has not occurred; using this option avoids frequent refining and coarsening with chattering contact. You can also specify the value MAT to refine only elements containing materials that are in contact with the Lagrangian surfaces. If no values are specified, MAT will be used except for materials with Mie-Grüneisen equations of state.
Refine elements in which significant plastic deformation occurs. Not supported for the critical state (clay) plasticity model.	PEEQ	Critical value of the equivalent plastic strain
Refine elements near a sharp density gradient	DENSITY	You can specify two values for this criterion. The first value is the critical value of the density gradient, computed as the ratio between the change of density across element faces and the density of the material inside the element; the second value is the critical density. For an element to be selected, both the density and the density gradient must exceed the critical value.
Refine elements near a sharp pressure gradient	PRESS	You can specify two values for this criterion. The first value is the critical value of the pressure gradient, computed as the ratio between the change of pressure across element faces and the pressure of the material inside the element; the second value is the critical pressure. For an element to be selected, both the pressure and the pressure gradient must exceed the critical value.

Input File Usage: **ADAPTIVE MESH REFINEMENT*,
refinement criteria label, *value of the criteria*

Contact

When adaptive mesh refinement is specified in an Eulerian section that is involved in general contact, dynamic seeding is activated by default. Dynamic seeding allows more seeds to be created once the Eulerian elements near a Lagrangian face are refined; these seeds will be deleted once these Eulerian elements are coarsened.

Dynamic seeding can also be useful for Eulerian analyses with no adaptive mesh refinement.

Input File Usage: **CONTACT CONTROLS ASSIGNMENT*, SEEDING=DYNAMIC

Limitations

You cannot apply adaptive mesh refinement on elements where Eulerian mesh motion is also applied.

Particle Methods

In this section:

- *Discrete Element Method*
- *Continuum Particle Analyses*
- *Particle Generator*
- *Lumped Kinetic Molecular Method*

Discrete Element Method

Products: Abaqus/Explicit Abaqus/Viewer

References:

- *Discrete Particle Elements*
- *Particle Generator*
- **DISCRETE SECTION*
- **DISCRETE ELASTICITY*
- **CONTACT*
- **INITIAL CONDITIONS*
- **PARTICLE GENERATOR*

Overview

The discrete element method (DEM):

- is intended for modeling events in which large numbers of discrete particles contact each other;
- models each particle with a single-node element that has a rigid spherical shape, which might represent an individual grain, tablet, shot peen, or other simple body;
- is a versatile tool for modeling particulate material behavior in pharmaceutical, chemical, food, ceramic, metallurgical, mining, and other industries; and
- is not meant for modeling deformation of a continuum, but DEM can be used together with finite elements for modeling discrete particles interacting with deformable continua or other rigid bodies.

Introduction

The discrete element method (DEM) is an intuitive method in which discrete particles collide with each other and with other surfaces during an explicit dynamic simulation. Typically, each DEM particle represents a separate grain, tablet, shot peen, etc. DEM is not applicable to situations in which individual particles undergo complex deformation. Therefore, DEM is unlike, and conceptually simpler than, the smoothed particle hydrodynamic (SPH) method in which groups of particles collectively model a continuum body (see *Smoothed Particle Hydrodynamics*).

For example, DEM is well-suited for particle mixing applications, such as that shown in *Figure 1*. In this application DEM is used to model the initially separated blue and white particles, and rigid finite elements are used to model two mixing augers and the box-shaped container. The sequence of deformed plots in *Figure 1* shows the particle response as the augers turn. DEM results for simulations such as this are often best viewed with animations. Another example of using DEM for a mixing application is described in *Mixing of granular media in a drum mixer*.

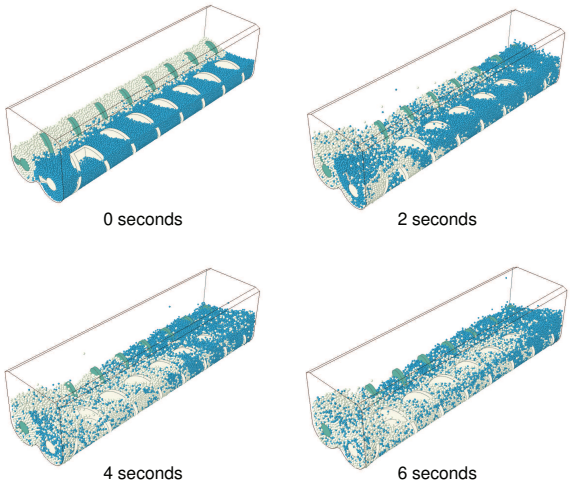


Figure 1: DEM particle mixing example.

Each DEM particle is modeled with a single-node element of type PD3D. These elements are rigid spheres with specified radii. Nodes of PD3D elements have displacement and rotational degrees of freedom. Rotations of DEM particles can significantly influence contact interactions when friction is considered.

General contact definitions are easily extended to include interactions among DEM particles and interactions between DEM particles and finite element-based surfaces. Large relative motion among particles is typical for DEM applications. Particle-to-particle interactions can involve like or unlike particles. Each particle can be involved in many contact interactions simultaneously. DEM particle interactions use finite contact stiffness, which introduces some compliance into the particle systems. For example, the contact stiffness can be specified such as to reflect the macroscopic stiffness of a packed granular material model with DEM.

For example, consider the interactions between the two spherical particles shown in *Figure 2*.

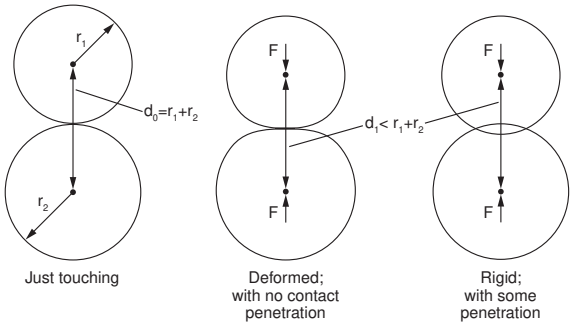


Figure 2: Interactions between spherical particles.

The three cases show two undeformed spheres just touching, two deformed spheres pushed toward one another with contact strictly enforced, and two rigid spheres pushed toward one another with some penetration. The distance between the centers of the spheres is the same for the cases shown in the middle and on the right in *Figure 2*. The physical behavior corresponds to the middle case. The case on the right corresponds to a DEM approximation. If the variable δ is defined as

$$\delta = r_1 + r_2 - d,$$

where r_1 and r_2 are the radii of the two spheres and d is the distance between the sphere centers, $\delta = 0$ when the undeformed spheres are just touching and $\delta > 0$ if the distance between the sphere centers is less than the combined radii. For the DEM approximation, δ corresponds to the maximum penetration distance between the particles. You can improve the accuracy for some DEM applications by tuning the contact stiffness relationship

(contact force F versus penetration δ) for DEM particles to reflect the Hertz contact solution (middle case in *Figure 2*). See *Mixing of granular media in a drum mixer* for further discussion of tuning the contact stiffness.

Applications

DEM is a versatile tool for modeling particulate material behavior in pharmaceutical, chemical, food, ceramic, metallurgical, mining, and other industries. DEM applications include the following categories:

Particle packing

involves processes such as pouring or deposition under gravity (such as sandpiling), vibration after deposition of particles, and compaction.

Particle flow

might occur under gravity only (as in the case of a hopper) or under gravity and other driving forces (such as for mixers and mills).

Particle-fluid interaction

occurs in transport of granular material within a fluid flow, during wavelike motion, and during fluidization (wherein fluid flows upward through a bed of particles).

DEM can provide insight for many situations that are difficult to investigate with other computational methods or with physical experiments.

Strategies for Creating and Initializing a DEM Model

Particulate media often consist of randomly distributed grains of varying sizes. Generating an initial mesh for a DEM analysis can be challenging. A common strategy for DEM is to specify approximate initial positions of particles with some gaps between them and to allow the particles to settle into position under gravity loading during the first step. For example, this strategy is used for the mixing analysis shown in *Figure 1*: the augers are kept stationary during the first step in which the particles are allowed to settle, and the augers are turned during the second step to study the mixing behavior (which is the focus of the figure shown). The particle generator can be used to create DEM models (see *Particle Generator*).

Strategy for Reducing Solution Noise

The solution noise generated by numerous opening and closing contact conditions can be reduced by applying a small amount of mass proportional damping. For more information, see *Alpha Damping*.

Time Incrementation Considerations for Non-Hertzian Contact

DEM uses the explicit dynamics procedure type. In most cases Abaqus/Explicit automatically controls the time increment size, as discussed in *Automatic Time Incrementation*, based on stiffness and mass characteristics of the model. The relationship between the maximum stable time increment size, mass, and stiffness properties is complex. The stable time increment size tends to be proportional to the square root of mass and inversely proportional to the square root of stiffness. However, a stable time increment cannot be computed for each PD3D element because the particles are rigid, so you must specify a fixed time increment size for purely DEM analyses (see *Fixed Time Incrementation*).

Contact interactions among DEM particles can influence the appropriate time increment size. DEM analyses without tightly packed particles might simply call for a contact stiffness that is large enough to avoid significant

penetrations, rather than a contact stiffness that is highly representative of physical stiffness characteristics of the particles (which are each modeled as rigid with DEM). If you do not specify the contact stiffness, Abaqus/Explicit assigns a default (penalty) contact stiffness based on the time increment size and mass/rotary inertia characteristics of the particles. This is undesirable since it is difficult to ensure that the time increment size is small enough to result in a sufficiently large penalty stiffness.

If you specify the non-Hertzian DEM contact stiffness, you must ensure that the time increment used for the analysis is small enough to avoid numerical instabilities. For dense three-dimensional packing of particles where each particle simultaneously contacts many particles, the numerical stability considerations are complex. A general guideline is that the time increment should not exceed $\mathbf{0.4}\sqrt{\frac{m}{k}}$, where m and k represent the particle mass and contact stiffness, respectively. In some applications an even smaller time increment, such as $\mathbf{0.1}\sqrt{\frac{m}{k}}$, might result in an improved solution.

If particle velocities become very large, the amount of incremental motion can influence the appropriate time increment size. Accurate resolution of particle motion sometimes requires specifying a smaller time increment than the maximum numerical stability time increment.

Automatic Time Incrementation for Hertzian Contact

The time increment size for a stable and accurate DEM analysis depends on several different factors, such as the underlying contact properties, the size of the particles, and the relative motion of the particles. Since these controlling factors are problem dependent and vary during the analysis, choosing an appropriate direct time increment for a DEM analysis can be difficult. When the Hertz or JKR-type pressure overclosure is specified, Abaqus/Explicit automatically controls the time increment size to achieve a stable and accurate solution.

The following criteria are used to control the time increment size:

- Stability: based on particle mass, inertia, contact stiffness, and the number of contact interactions with other particles
- Tracking accuracy: based on particle size and velocity
- Duration of collision: restricted to a fraction of the duration of the collision between impacting particles
- Rayleigh wave propagation: based on the time taken for a Rayleigh wave to travel from pole to pole of a particle
- Separation distance for JKR model: limit the relative normal motion based on the separation distance for the JKR-type pressure overclosure
- Tangential tracking accuracy: limit the relative tangential slip between particles. The slip depends on the particle rotation.

The chosen time increment is the smallest of the above six criterion. A scaling factor is associated with each of the above criterion.

Depending on the nature of the analysis, different criterion might control the time increment size at different times during the analysis. The stability criterion might dictate the time increment when particles are confined and subjected to compression, whereas for fast-moving impacting particles, the duration of collision criterion might control the time increment. The default scaling factor applied to each of the above criterion is based on numerical experimentation for low- to moderate-speed collisions. For high-speed impacts where the contact stiffness increases rapidly, the scaling factors related to the Rayleigh wave and collision duration might need to be reduced. You can specify values for the scaling factors using section controls.

The automatic time incrementation also works in conjunction with the particle generator provided the contact interactions specify the Hertz or JKR-type pressure overclosure type.

Input File Usage: Use the following options to modify the default scaling factors for one or more criterion for automatic time incrementation:

**DISCRETE SECTION*, CONTROLS=*section_control_name*
…
**DISCRETE ELASTICITY*
effective Young's Modulus, *effective Poisson's ratio*

**SECTION CONTROLS*, NAME=*section_control_name*
blank line
blank line
blank line
blank line

$f_{stability}$, $f_{tracking}$, $f_{collision}$, $f_{Rayleigh}$, $f_{separation_dist}$, $f_{rotation}$

**STEP*
**DYNAMIC*, EXPLICIT, ELEMENT BY ELEMENT
…

Initial Conditions

Initial conditions pertinent to mechanical analyses can be used in a discrete element method analysis. All of the initial conditions that are available for an explicit dynamic analysis are described in *Initial Conditions*.

Boundary Conditions

Boundary conditions are defined as described in *Boundary Conditions*. Boundary conditions are rarely applied on individual particles in DEM.

Loads

The loading types available for an explicit dynamic analysis are explained in *About Loads*. Gravity loads are very important for settling and particulate flow analysis in DEM. Concentrated loads are rarely applied on particles.

Elements

The discrete element method uses PD3D elements to model individual particles. These 1-node elements define individual grains of a particulate media, are spherical in shape, and are modeled as rigid (any compliance is built into the contact model). These particle elements use existing Abaqus functionality to reference element-related features such as initial conditions, distributed loads, and visualization. You can define these elements in a similar fashion as you would define point masses or rotary inertia. The coordinates of the node of a particle correspond to the center location of the physical grain of material. PD3D elements are assigned to a discrete section definition, where particle characteristics are specified. For more information, see *Discrete Particle Elements*.

A particle tracking box is established at the beginning of the analysis to define the rectangular region within which the particle search (finding all neighbors for all particles) is performed. You can modify the size of the box as discussed in *Using Section Controls to Define the Particle Tracking Box for DEM and SPH Particles*.

Input File Usage: Use the following options to define a discrete element medium:

**ELEMENT*, TYPE=PD3D, ELSET=*particle_body*
element number, *node number*

**DISCRETE SECTION*, ELSET=*element_set_name*

Particle Swelling

Granular materials swell due to absorption of fluids or through complex electrochemical interactions with the surrounding medium. Properties such as the porosity of the granular media depend on the particle size. Therefore, it is important to capture the swelling of particles in order to study the changes in media porosity. In batteries, the swelling of particles can be either reversible or irreversible due to the lithiation and delithiation process during a charge and discharge cycle. For reversible swelling, the size of the particle can go back to its original volume. For irreversible swelling, the size increase is permanent. The aging process in batteries is characterized by irreversible swelling that is a surface deposition process.

Reversible Swelling

Abaqus/Explicit computes the reversible change in size of the particle from the original particle size r_0 based on either the specified volume change ratio or the specified radius change ratio. The volume change ratio is defined as

$$v' = \frac{v - v_0}{v_0},$$

where v_0 is the original particle volume and v is its current volume. The radius change ratio is defined as

$$r' = \frac{r - r_0}{r_0},$$

where r is its current radius.

To apply reversible swelling, specify either v' or r' as a function of temperature and/or field variables.

Input File Usage: Use the following options to specify particle swelling for a set of discrete particles as a function of temperature and field variables:

**DISCRETE SECTION*, PARTICLE BEHAVIOR=*particle_behavior_name*
…
**PARTICLE BEHAVIOR*, NAME=*particle_behavior_name*
Provide the particle swelling and/or particle surface deposition as suboptions of the particle behavior.
For a given particle behavior, you can specify at most one particle swelling definition and one particle surface deposition definition.

Use the following option to define particle swelling by specifying the volume change ratio as a function of temperature and field variables.

**PARTICLE SWELLING*, DEFINITION=*volumetric*
v', T, $FV1$,...

Use the following option to define particle swelling by specifying the radius change ratio as a function of temperature and field variables.

**PARTICLE SWELLING*, DEFINITION=*radial*
r', T, $FV1$,...

Surface Deposition

Surface deposition is an irreversible swelling process. You can specify the surface deposition in terms of

$$SD^{surf} = \frac{\Delta r}{r^0},$$

where SD^{surf} is the ratio of the thickness of deposited material, Δr to the original radius r^0. Alternatively, you can specify the thickness of the deposited material, Δr, directly. Either specification method supports dependencies on temperature and/or field variables.

Abaqus/Explicit maintains the irreversibility of the surface definition by retaining the maximum deposited thickness at any given increment even if the specified surface deposition reduces during the next increment.

Input File Usage: Use the following options to define surface deposition for a set of discrete particles as a function of temperature and field variables.

**DISCRETE SECTION*, PARTICLE BEHAVIOR=*particle_behavior_name*
…
**PARTICLE BEHAVIOR*, NAME=*particle_behavior_name*
Provide the particle swelling and/or particle surface deposition as suboptions of the particle behavior.
For a given particle behavior, you can specify at most one particle swelling definition and one particle surface deposition definition.

Use the following option to define particle surface deposition by specifying the ratio of the thickness of the deposited material to the original particle radius as a function of temperature and field variables.

**PARTICLE SURFACE DEPOSITION*, DEFINITION=*ratio*
SD^{surf}, T, $FV1$,...

Use the following option to define particle surface deposition by specifying the thickness of the deposited material as a function of temperature and field variables.

**PARTICLE SURFACE DEPOSITION*, DEFINITION=*thickness*
Δr, T, $FV1$,...

Modeling Particle Swelling

When modeling particle swelling:

- You must exercise caution when you use the particle swelling feature, because if the swelling is too rapid, the analysis might become unstable.
- You can use amplitude definitions to change the temperature and/or field variables to control the reversible and irreversible particle swelling/deposition.
- For a given particle behavior, you can specify at most one particle swelling definition and one particle surface deposition definition.
- Discrete particles are rigid; therefore, you cannot apply thermal expansion. To obtain the same increase in a discrete particle size via swelling as that obtained via modeling the particle as a corresponding meshed deformable spherical body with thermal expansion, specify tabular values as described below. If α is the coefficient of thermal expansion of a deformable body and the change in temperature is ΔT, set the volume or radius change ratio as follows:

	Volume Change Ratio	Radius Change Ratio
Geometrically linear deformable analysis	$(1+\alpha\Delta T)^3-1$	$\alpha\Delta T$
Deformable analysis that accounts for geometric nonlinearity	$e^{3\alpha\Delta T}-1$	$e^{\alpha\Delta T}-1$

Constraints

Since PD3D elements are Lagrangian elements, their nodes can be involved in other features such as constraints. Although the PD3D element has a spherical shape, it is possible to model grains of complex shapes by clustering particles together, as illustrated in *Figure 3*. A cluster is a group of particles that are held together either rigidly or via compliant connections.

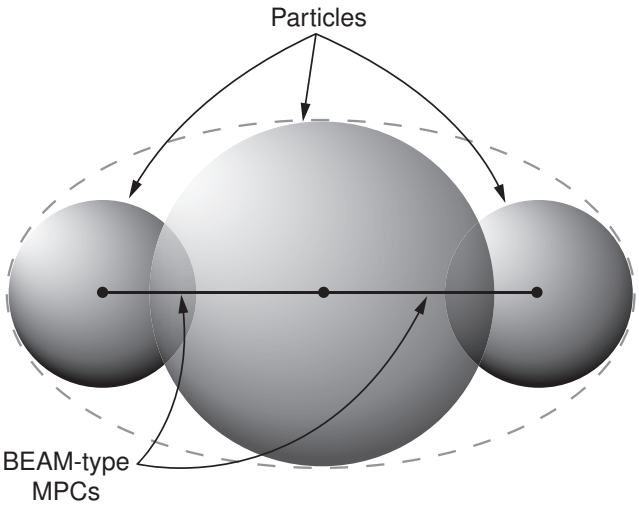


Figure 3: Rigid cluster of particles.

The particles in a cluster might overlap with each other. Contact forces that try to push apart overlapping particles of a cluster will exist unless contact exclusions are specified among particles of a cluster (see *Specifying Contact Exclusions*). These contact forces will have no effect on particles held together rigidly.

The particle-cluster approach might not replicate the precise geometry of actual grains. For example, the cluster shown in *Figure 3* might approximate an ellipsoidal shape (indicated by the dashed line in the figure). More spherical particles of various sizes can be added to the cluster to obtain a closer approximation of the true shape.

Define BEAM-type multi-point constraints between nodes of a group of particles to create a rigid cluster. Clusters of overlapping particles that do not involve multi-point constraints might exhibit nonphysical behavior. For more information, see *General Multi-Point Constraints*.

General Particle Clustering

General particle clustering allows you to create arbitrary grain shapes without having to use multi-point constraints. One of the main advantages of the general particle clustering approach is that it can be executed in domain parallel mode. (Particle clustering using *General Multi-Point Constraints* is available only for serial execution.)

Tip: You can generate a section definition that approximates any arbitrarily shaped body for a discrete particle analysis. For more information, see Dassault Systèmes Knowledge Base item QA000000369419 (*https://support.3ds.com/knowledge-base/?q=docid:QA00000369419*).

Defining particle clusters

A particle cluster is formed by combining particles in a rigid fashion. The particles in a cluster might or might not overlap with other particles as shown in *Figure 4*. A particle cluster consists of a parent particle and one or more child particles as shown in *Figure 5*.

A parent particle is always a user-defined PD3D element. You should group all parent particles that have the same cluster geometry in an element set and then assign that element set of parent particles to a discrete section.

You do not need to define the child particles associated with a parent particle like regular particles. Abaqus/Explicit automatically creates child particles based on the location and size of the child particle. You can specify the position and size of a child particle as part of the discrete section definition. To define the position of the child, you must specify the offset vector from the center of the parent particle to the center of the child particle and the orientation of this offset vector in the local axes system centered at the parent particle. *Figure 6* shows the offset vector. You specify the following as a fraction of the parent particle radius:

- the magnitude of the offset vector from the center of the parent particle to the center of the child particle, and
- the radius of the child particle.

You specify the orientation of the offset vector using the two spherical angles in the local coordinate system centered at the parent particle as shown in *Figure 6*.

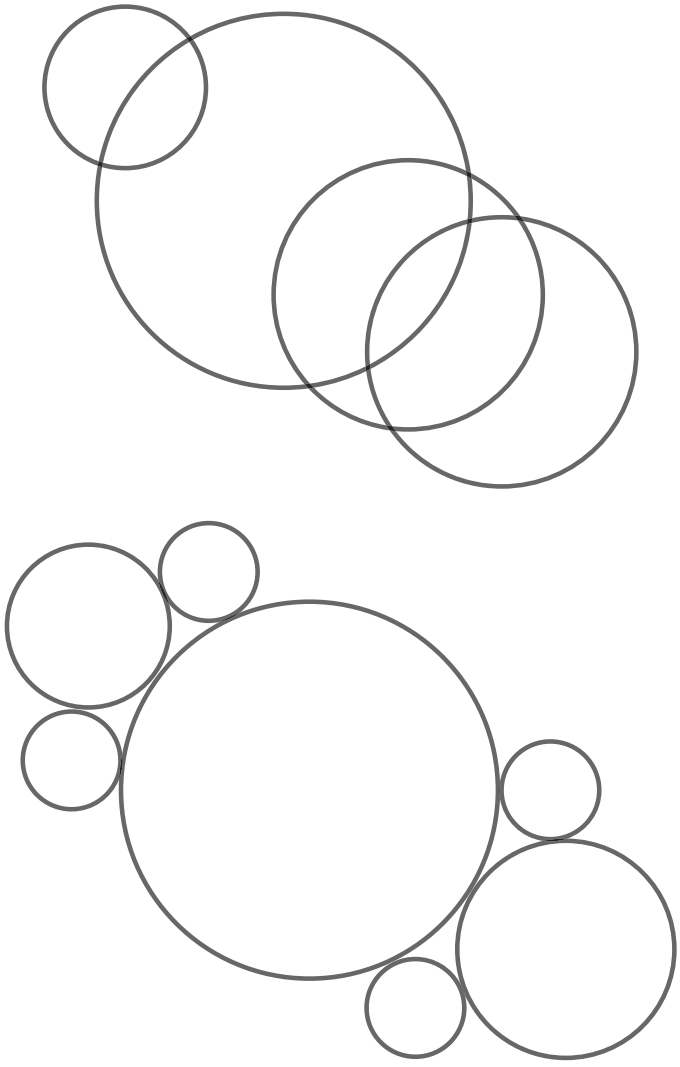


Figure 4: Overlapping and nonoverlapping clusters.

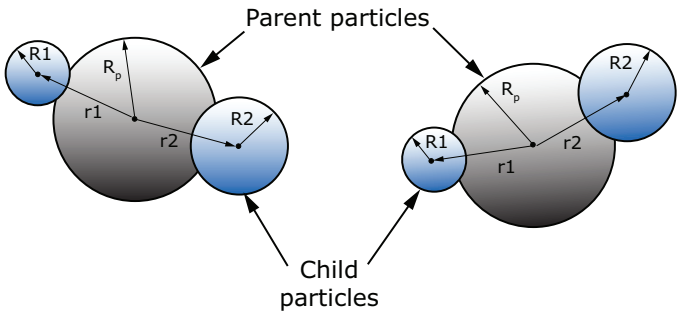


Figure 5: Parent and child particles in a cluster.

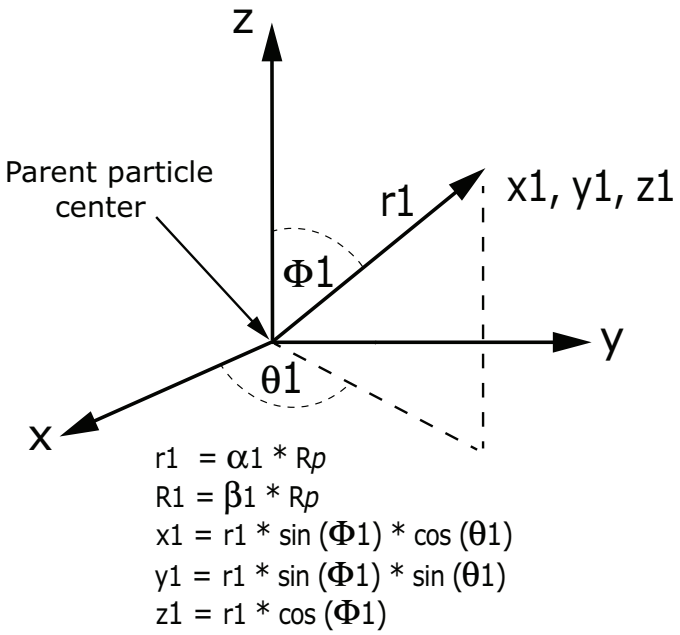


Figure 6: Particle cluster definition.

Input File Usage: Use the following options to define a particle cluster:

**DISCRETE SECTION*, ELSET=*element_set_name*

R_p

$\alpha 1, \beta 1, \theta 1, \varphi 1$

$\alpha 2, \beta 2, \theta 2, \varphi 2$

$\dots$

Defining mass and inertia for clusters

You must specify the mass and inertia of every particle cluster. Since the cluster geometry definition is specified in terms of the parent particle radius, R_p, the mass and inertia of the cluster are functions of the parent particle radius. Define a table of cluster mass and inertia values for different values of the parent radius. Abaqus/Explicit interpolates the mass and inertia components of the cluster using the radius of the parent particle.

Each child particle in the cluster inherits the contact properties of the parent particle. Therefore, you need to provide contact definitions only for the surface spanning the parent particles. Abaqus/Explicit automatically

creates the required contact definitions for the child particles based on the contact interactions defined for the parent particles. Abaqus/Explicit automatically excludes contact between the parent and child particles.

Input File Usage: Use the options below to define a particle cluster. To associate the cluster mass and inertia table with the parent particle set, specify the table as a suboption of the discrete element section specification.

**DISCRETE SECTION*, ELSET=*element_set_name*

R_p

$\alpha 1$, $\beta 1$, $\theta 1$, $\varphi 1$

...

**CLUSTER MASS INERTIA TABLE*

$M1$, $I1XX$, $I1XY$, $I1XZ$, $I1YY$, $I1YZ$, $I1ZZ$, $R1$

$M2$, $I2XX$, $I2XY$, $I2XZ$, $I2YY$, $I2YZ$, $I2ZZ$, $R2$

...

Interactions

Contact is an essential ingredient for DEM analyses, as discussed above. General contact is used to define contact involving DEM particles. A DEM particle can be involved in multiple contact interactions simultaneously with

- another particle with the same discrete section definition;
- another particle with a different discrete section definition; and
- a surface based on finite elements.

Modeling contact between DEM particles requires that the particles be explicitly included in general contact as element-based surfaces using contact inclusions (see *Element-Based Surface Definition*). See *About General Contact in Abaqus/Explicit* for a discussion of general contact. By default, the particles are not part of the general contact domain, similar to other 1-node elements (such as point masses).

Contact stiffness for DEM is often intended to account for the physical stiffness characteristics of the particles because DEM models each particle as rigid; therefore, nondefault contact property assignments are common for DEM interactions.

Normal and Tangential Contact Forces

Figure 7 is a schematic representation of the contact stiffness and damping between two particles. The spring stiffness K_n acts in the contact normal direction and might represent a simple linear or a nonlinear contact stiffness. The dashpot C_n represents contact damping in the normal direction. The tangential spring stiffness K_t along with the friction coefficient μ represent friction between the particles. The dashpot C_t represents contact damping in the tangential direction.

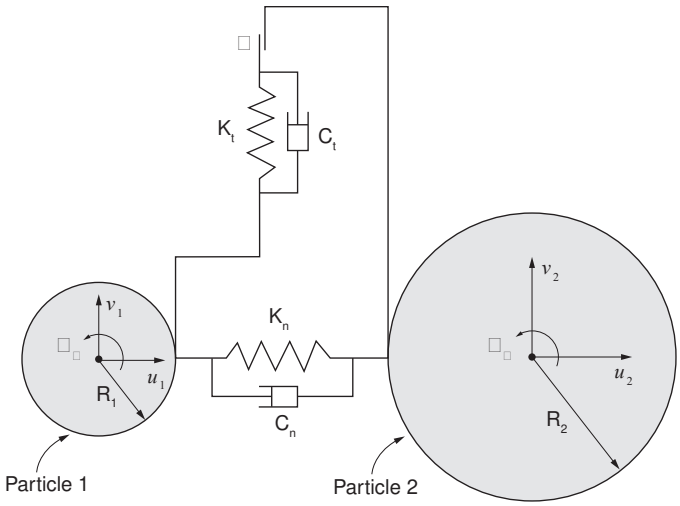


Figure 7: Normal and tangential contact interaction between two discrete elements.

Figure 7 shows that the tangential contact forces acting on particle surfaces cause moments at particle centers. Interactions involving DEM particles account for moment transfer across the interface.

Hertz Normal Contact between Particles

The Hertz elastic solution relating contact force, F, to the approach distance, δ, for two contacting spherical particles is:

$$F = \frac{4}{3} E^* \sqrt{R} \sqrt{\delta^3},$$

where

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

and

$$\frac{1}{E^*} = \frac{1-\nu_1^2}{E_1} + \frac{1-\nu_2^2}{E_2}.$$

E_1, ν_1 and E_2, ν_2 are the effective Young's modulus and Poisson's ratio of the two contacting particles, respectively. R_1 and R_2 are the radii of the two contacting particles, respectively. The normal contact stiffness is $K_n = 2E^* \sqrt{R} \sqrt{\delta}$. You must specify the effective Young's modulus and Poisson's ratio for a contacting particle. In addition, you must specify the Hertz-type pressure overclosure.

K_{max} is the limiting value of the normal contact stiffness, K_n, beyond which the normal contact force increases linearly using the slope of the K–δ curve at K_{max}.

Input File Usage: Use the following options to define Hertz-type normal contact behavior:

**DISCRETE SECTION*, ELSET=*element_set_name*
**DISCRETE ELASTICITY*
effective Young's Modulus, *effective Poisson's ratio*
...
**SURFACE INTERACTION*
**SURFACE BEHAVIOR*, PRESSURE-OVERCLOSURE=HERTZ
K_{max}

Johnson-Kendall-Roberts Adhesive Normal Contact between Particles

The JKR model relates contact force, F, to the radius, a, of the contact area, as follows:

$$F = \frac{4E^*a^3}{3R} - \sqrt{8\pi\Gamma E^* a^3}.$$

The approach distance, δ, for two contacting spherical particles is related to the radius, a, of the contact area, as follows:

$$\delta = \frac{a^2}{R} - \sqrt{\frac{2\pi\Gamma a}{E^*}}.$$

In the above equations

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

and

$$\frac{1}{E^*} = \frac{1-\nu_1^2}{E_1} + \frac{1-\nu_2^2}{E_2}.$$

In the above equations Γ is the surface energy per unit area of the two contacting particles. Like nonadhesive Hertz contact E_1, ν_1 and E_2, ν_2 are the effective Young's modulus and Poisson's ratio of the two contacting particles, respectively. R_1 and R_2 are the radii of the two contacting particles, respectively. You must specify the effective Young's modulus and Poisson's ratio for a contacting particle. In addition, you must specify the JKR-type pressure overclosure.

K_{max} is the limiting value of the normal contact stiffness, beyond which the normal contact force increases linearly.

Figure 8 shows the force-displacement curve for the JKR adhesion model. Adhesion between particles is triggered on first contact. The pull-off force, $F_c = -\frac{3}{2}\pi\Gamma R$, is the maximum tensile force. Particles continue to experience adhesion force even after physical separation occurs. The adhesion force between two particles goes to zero at a separation distance of

$$\delta_{separation} = -\frac{3}{4}\left(\frac{\pi^2\Gamma^2 R}{E^{*2}}\right)^{\frac{1}{3}}.$$

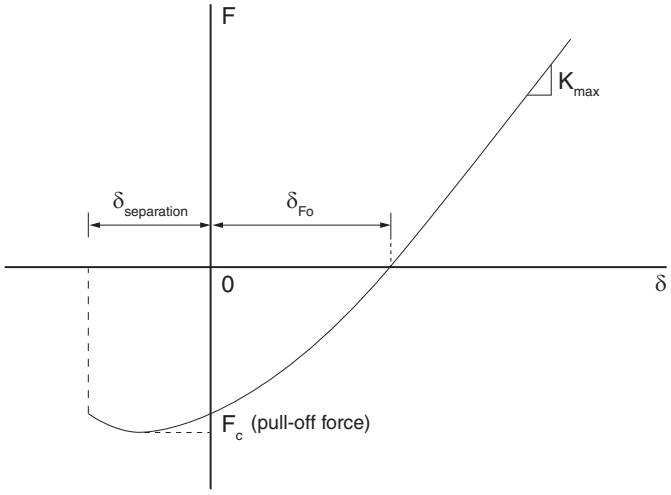


Figure 8: Force versus approach distance relation for the JKR model.

It can be seen from *Figure 8* that adhesive forces are nonzero at $\delta = 0$ and reduce to zero at $\delta = \delta_{F0}$. In some situations it might be desirable to have zero adhesive forces when $\delta = 0$. You can achieve this by applying a horizontal shift of $\delta = \delta_{F0}$ to the force-displacement curve shown in *Figure 8*. *Figure 9* shows the shifted curve. The pull-off force remains unchanged due to the shift, whereas the new separation distance $\delta_{separation_shifted} = \delta_{separation} + \delta_{F0}$. Abaqus automatically computes the horizontal shift δ_{F0} for the "shifted" JKR type adhesive behavior.

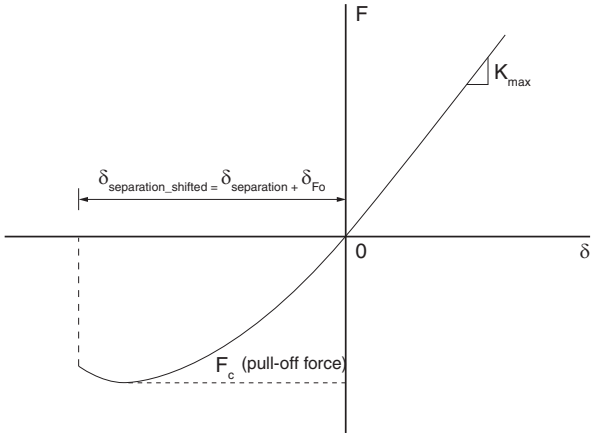


Figure 9: Force versus approach distance relation for the shifted JKR model.

Input File Usage: Use the following options to define JKR-type normal contact behavior:

**DISCRETE SECTION*, ELSET=*element_set_name*
**DISCRETE ELASTICITY*
effective Young's modulus, *effective Poisson's ratio*
...
**SURFACE INTERACTION*
**SURFACE BEHAVIOR*, PRESSURE-OVERCLOSURE=JKR
Surface energy per unit area, K_{max}

Use the following options to define the shifted JKR-type normal contact behavior:

**DISCRETE SECTION*, ELSET=*element_set_name*
**DISCRETE ELASTICITY*
effective Young's modulus, *effective Poisson's ratio*
...
**SURFACE INTERACTION*
**SURFACE BEHAVIOR*, PRESSURE-OVERCLOSURE=JKR
Surface energy per unit area, K_{max}, *SHIFTED*

Output

No element output is available for PD3D elements. The nodal output includes all output variables generally available in Abaqus/Explicit analyses (see *Abaqus/Explicit Output Variable Identifiers*).

You can visualize particles in Abaqus/CAE by doing the following:

1. Click the **Entity Display** tab.

2. From **View->ODB Display Options**, select **Render element particle edges** .

The display visualization changes depending on the mode or plot type as described in the table below.

Mode/Plot type	Visualization Display
Wireframe	Circular ring for each particle element
Hidden	Combination of a circular ring and disc for each particle element
Filled	Combination of a circular ring and disc for each particle element
Shaded	Combination of a circular ring and sphere for each particle element
Contour	Combination of a circular rings and colored sphere for field output variable values

Limitations

Discrete element method analyses are subject to the following limitations:

- Volume average output for stress, strain, and other similar continuum element output is not available for DEM analysis.
- Only a spherical shape is supported for PD3D elements.
- PD3D elements cannot be part of a rigid body definition.
- Models with rigid clusters using PD3D elements and BEAM-type multi-point constraints cannot be run in domain parallel mode.
- It is not possible to specify cohesive or thermal contact between PD3D elements or between PD3D elements and other elements.
- Rolling friction is ignored for contact between PD3D elements or between PD3D elements and other elements.
- User-defined surface interaction is not supported for contact between PD3D elements.
- Although supported in Abaqus/Viewer, the functionality is not supported in Abaqus/CAE. You can use the existing functionality in Abaqus/CAE to generate mass elements, write an input file, and then manually edit the input file to convert the mass elements to particles. Alternatively, you can create a mesh using C3D8R elements, write an input file, and then use a script to convert these elements to particles. An example of a Python script to convert solid elements to SPH particles is described in “Generating SPH particle elements from a solid mesh” in the Dassault Systèmes Knowledge Base at *http://support.3ds.com/knowledge-base/*.

DEM computations are distributed across parallel domains except when multiple discrete sections are defined with different alpha damping parameters (which will degrade parallel scalability). DEM analyses are subject to the following limitations if multiple CPUs are used:

- Contact output is not supported for DEM secondary nodes.
- Energy history output other than for the whole model is not supported.
- Dynamic load balancing cannot be activated.
- If any DEM particles participate in general contact, all DEM particles must be included in the general contact definition.
- At least 10,000 DEM particles per domain is suggested to achieve good scalability.
- A significant increase in memory usage might be needed if a large number of CPUs is used.

Input File Template

The following example illustrates a discrete element method analysis:

```
*HEADING
…
*ELEMENT, TYPE=PD3D, ELSET=name
Element number, node number
…
*DISCRETE SECTION, ELSET=name
Particle radius
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
*NSET, NSET=name, ELSET=name
*SURFACE, NAME=name
,
**
*SURFACE INTERACTION
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
Data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
*CONTACT PROPERTY ASSIGNMENT
**
*CONTACT CONTROLS ASSIGNMENT
*OUTPUT, FIELD
*END STEP
```

References

- Cundall, P. A., and O. D. Strack, “A Distinct Element Method for Granular Assemblies,” *Geotechnique*, vol. 29, pp. 47–65, 1979.
- Johnson, K. L., K. Kendall, and A. D. Roberts, “Surface Energy and the Contact of Elastic Solids,” *Proceedings of the Royal Society of London*, vol. 324, pp. 301–313, 1971.
- Munjiza, A., and K. R. F. Andrews, “NBS Contact Detection Algorithm for Bodies of Similar Size,” *International Journal for Numerical Methods in Engineering*, vol. 43, pp. 131–149, 1998.
- O'Sullivan, C., and J. D. Bray, “Selecting a Suitable Time Step for Discrete Element Simulations that Use the Central Difference Time Integration Scheme,” *Engineering Computations*, vol. 21(2/3/4), pp. 278–303, 2004.
- Zhu, H. P., Z. Y. Zhou, R. Y. Yang, and A. B. Yu, “Discrete Particle Simulation of Particulate Systems: A Review of Major Applications and Findings,” *Chemical Engineering Science*, vol. 63, pp. 5728–5770, 2008.
- Zhu, H. P., Z. Y. Zhou, R. Y. Yang, and A. B. Yu, “Discrete Particle Simulation of Particulate Systems: Theoretical Developments,” *Chemical Engineering Science*, vol. 62, pp. 3378–3396, 2007.

Continuum Particle Analyses

In this section:

- *Smoothed Particle Hydrodynamics*
- *Finite Element Conversion to SPH Particles*

Smoothed Particle Hydrodynamics

Products: Abaqus/Explicit

References:

- *Continuum Particle Elements*
- **SOLID SECTION*
- **SECTION CONTROLS*
- **INITIAL CONDITIONS*

Overview

Smoothed particle hydrodynamics (SPH) is a numerical method that is part of the larger family of meshless (or mesh-free) methods.
For these methods you do not define nodes and elements as you would normally define in a finite element analysis; instead, only a collection of points are necessary to represent a given body. In smoothed particle hydrodynamics these nodes are commonly referred to as particles or pseudo-particles.

Introduction

The example shown in *Figure 1* contrasts the two approaches. Both discrete representations model the initial configuration of a body of fluid inside a bottle, as described in detail in *Impact of a water-filled bottle*. The model on the left is a traditional tetrahedron mesh of the volume occupied by the fluid. On the right, the same volume is represented by a collection of discrete points. In the latter case, there are no edges connecting the points as these points (pseudo-particles) do not require the definition of multiple-node element connectivity, as is the case in the traditional finite element representation on the left. An alternative to directly defining particle elements is to define conventional continuum finite elements and automatically convert them to particle elements at the beginning of the analysis or during the analysis, as discussed in *Finite Element Conversion to SPH Particles*.

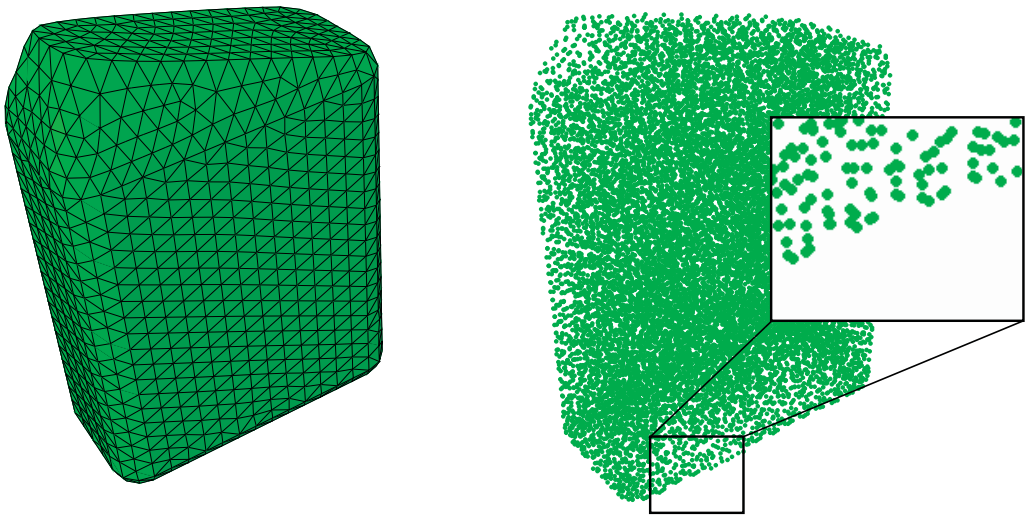


Figure 1: Finite element mesh and SPH particle distribution.

Smoothed particle hydrodynamics is a fully Lagrangian modeling scheme permitting the discretization of a prescribed set of continuum equations by interpolating the properties directly at a discrete set of points distributed over the solution domain without the need to define a spatial mesh. The method's Lagrangian nature, associated with the absence of a fixed mesh, is its main strength. Difficulties associated with fluid flow and structural problems involving large deformations and free surfaces are resolved in a relatively natural way.

At its core, the method is not based on discrete particles (spheres) colliding with each other in compression or exhibiting cohesive-like behavior in tension as the word particle might suggest. Rather, it is simply a clever discretization method of continuum partial differential equations. In that respect, smoothed particle hydrodynamics is quite similar to the finite element method. SPH uses an evolving interpolation scheme to approximate a field variable at any point in a domain. The value of a variable at a particle of interest can be approximated by summing the contributions from a set of neighboring particles, denoted by the subscript *j*:

$$\langle f(\mathbf{r})\rangle \simeq \sum_j \frac{m_j}{\rho_j} f_j W(\mathbf{r}-\mathbf{r}_j, h),$$

Equation 1

where f_j , m_j , and ρ_j are the variable value, mass, and density of the *j* particle, and the summation over *j* includes all neighboring particles within the smoothing kernel domain. *Equation 1* represents the SPH discrete approximation to the continuous field, and it is the basis of all SPH formalisms.

The interpolating kernel function, *W*, with the characteristic width, *h* , (known as the smoothing length) is a generalization of the delta function subject to the unity normalization. A cubic spline kernel (default in Abaqus/Explicit) is most commonly used:

$$W(\mathbf{r}-\mathbf{r}_j, h) = \frac{1}{\pi h^3}\begin{Bmatrix} 1-\frac{3}{2}q^2+\frac{3}{4}q^3 & 0\le q\le 1 \\ \frac{1}{4}(2-q)^3 & 1<q\le 2 \\ 0 & q>2 \end{Bmatrix},$$

where

$$q = \frac{|\mathbf{r}-\mathbf{r}_j|}{h}.$$

An illustration of the kernel function is shown in *Figure 2*. The smoothing length, *h*, determines how many neighboring particles influence the interpolation for a particular point.

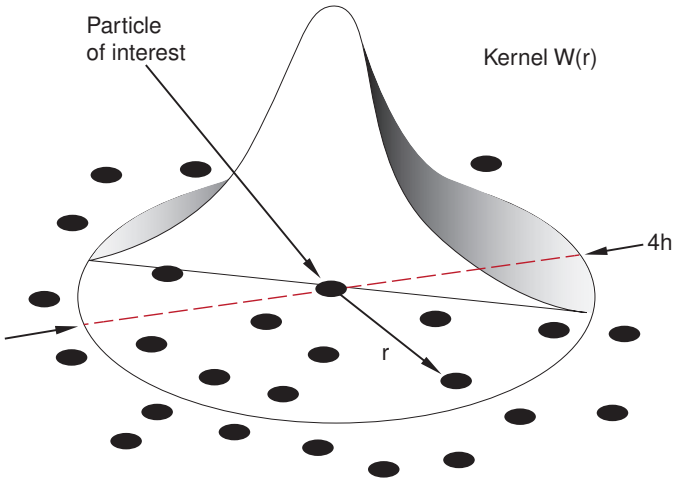


Figure 2: Kernel function.

The continuity and momentum equations for a particle of interest, *i*, can be written as:

$$\frac{d\rho_i}{dt} = \sum_j m_j\left(\mathbf{v}_i-\mathbf{v}_j\right)\cdot\frac{\partial}{\partial \mathbf{r}_i}W(\mathbf{r}_i-\mathbf{r}_j, h),$$

$$\frac{d\mathbf{v}_i}{dt} = -\sum_j m_j \left(\frac{\boldsymbol{\sigma}_j}{\rho_j^2} + \frac{\boldsymbol{\sigma}_i}{\rho_i^2} + \boldsymbol{\Pi}_{ij} \right) \cdot \frac{\partial}{\partial \mathbf{x}_i} W(\mathbf{x}_i - \mathbf{x}_j, h),$$

Equation 2

where $\mathbf{v}_j$ and $\boldsymbol{\sigma}_j$ are, respectively, the velocity and stress tensor of the particle *j*. Term $\boldsymbol{\Pi}_{ij}$ represents the contribution of artificial viscosity to the momentum equation (see *Artificial Viscosity*).

The SPH method has received substantial attention since its inception (*Gingold and Monaghan, 1977*), and the number of publications related to the method is now very large. A number of references are listed below.

The method can be used in combination with any of the material models available in Abaqus/Explicit (including user materials). You can specify initial conditions and boundary conditions as for any other Lagrangian model. Contact interactions with other Lagrangian bodies are also allowed, thus expanding the range of applications for which this method can be used.

In general, the method is less accurate than Lagrangian finite element analyses when the deformation is not too severe and than coupled Eulerian-Lagrangian analyses in higher deformation regimes.

Applications

Smoothed particle hydrodynamic analyses are effective for applications involving extreme deformation. Fluid sloshing, wave engineering, ballistics, spraying (as in paint spraying), gas flow, and obliteration and fragmentation followed by secondary impacts are a few examples. There are many applications for which both the coupled Eulerian-Lagrangian and the smoothed particle hydrodynamic methods can be used. In many coupled Eulerian-Lagrangian analyses the material to void ratio is small and, consequently, the computational effort might be prohibitively high. In these cases, the smoothed particle hydrodynamic method is preferred. For example, tracking fragments from primary impacts through a large volume until secondary impact occurs can be very expensive in a coupled Eulerian-Lagrangian analysis but comes at no additional cost in a smoothed particle hydrodynamic analysis.

Impact of a water-filled bottle includes an example of using the smoothed particle hydrodynamic method to model the violent sloshing associated with the impact.

Artificial Viscosity

Artificial viscosity in smoothed particle hydrodynamics is used to reduce unphysical high frequency oscillations in the computed response. Similar to other Lagrangian elements, SPH uses the sum of linear and quadratic viscous contributions. In *Equation 2* the artificial viscosity term $\boldsymbol{\Pi}_{ij}$ is defined as

$$\boldsymbol{\Pi}_{ij} = \begin{cases} \frac{-\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\bar{\rho}_{ij}} & (\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j) < 0 \\ 0 & (\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j) \geq 0 \end{cases},$$

where

$$\mu_{ij} = \frac{h(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)}{r_{ij}^2 + 0.01h^2}.$$

$\bar{c}_{ij}$ is the average sound speed of particle *i* and *j*, and r_{ij} is the distance between particles *i* and *j*.

The above artificial viscosity formulation applies uniformly to all SPH particles and is the default option. However, this option can possibly induce excessive shear viscous forces. To reduce the induced viscosity in shear motion, we adopt the approach proposed by *Balsara,1995* and multiply the artificial viscosity term $\boldsymbol{\Pi}_{ij}$ by a factor $f_{ij} = (f_i + f_j)/2$, where f_i is defined on each particle as

$$f_i = \frac{\left|\frac{\partial}{\partial \mathbf{x}_i}\cdot\mathbf{v}_i\right|}{\left|\frac{\partial}{\partial \mathbf{x}_i}\cdot\mathbf{v}_i\right| + \left|\frac{\partial}{\partial \mathbf{x}_i}\times\mathbf{v}_i\right|}.$$

The factor is evaluated only if the divergence of the velocity field is greater than a numerical threshold; otherwise, it is zero. This factor helps reduce the excessive shear viscous forces in shear dominated flows, while remaining unaffected in the case of normal shocks.

The linear and quadratic bulk viscosity coefficients, α and β, have default values of 0.06 and 1.2, respectively. When using SPH to model low viscosity fluids (such as water), the default linear bulk viscosity coefficient of 0.06 sometimes results in excessive viscous forces that can lead to less fluid-like behavior. In such cases, a smaller value of the linear bulk viscosity coefficient is recommended. You can control the amount of artificial viscosity included in a SPH analysis.

Input File Usage: Use the following option to specify a uniform bulk viscosity for all SPH particles:

**SECTION CONTROLS, SPH BULK VISCOSITY=UNIFORM*

Use the following option to specify a vorticity weighted bulk viscosity for SPH particles in shear dominated flows:

**SECTION CONTROLS, SPH BULK VISCOSITY=VORTICITY WEIGHTED*

SPH Tensile Instability Control

The classical SPH formulation can display a tensile instability that results in an unphysical clustering of particles. The clustering is particularly noticeable when materials are in a tensile stress state. To relieve the SPH tensile instability, Abaqus uses the artificial stress approach of *Monaghan, 2000*. In this method an artificial stress contribution that acts as a short-range repulsion force is added to mitigate the tensile instability. This repulsion force is used to prevent particles from getting too close to each other. This force is related to the inter-particle distance: the smaller the distance the greater the force. This feature is effective only when an initially uniform distribution of particles is used.

Input File Usage: Use the following option to specify the SPH tensile instability control:

**SECTION CONTROLS*, SPH TENSILE INSTABILITY CONTROL=YES

SPH Particle Generator

Smoothed particle hydrodynamic analysis uses the same particle generator as used to define discrete particles (PD3D, see *Particle Generator*). The SPH-type particle generator is a tool for automatically introducing SPH particles into the flow domain during the course of the analysis and is needed when introducing an SPH inflow boundary condition. To define the SPH particle generator, you must also specify the maximum number of particles that can be generated during the analysis. In addition, you must define an inlet surface associated with the SPH particle generator. If there is a user-defined PC3D element class in the model, it must have the same material property as the element class associated with the SPH particle generator.

The inlet surface can be thought of as an opening through which the particles flow into the flow domain. Similar to the PD3D particle generator, you must define an element-based surface for the inlet surface.



Note: A rigid surface cannot be used as the SPH particle generator inlet surface.

Care should be taken with respect to the positive surface normal direction at the inlet because the initial particle velocity is in this direction. All inlet surfaces associated with an SPH particle generator must be in one plane

(that is, have the same surface normal). If there is no user-defined PC3D element set in the model, SPH particles are generated uniformly on the inlet surfaces with the inter-particle distance defined as two times the particle thickness. If there is a user-defined PC3D element set in the model, SPH particles are generated on the inlet surfaces according to the spatial location of the user-defined SPH particles adjacent to the inlet surfaces. In this case if there are initially no SPH particles within a distance of two times the particle thickness to the inlet surfaces, the particle generation will not proceed. It is also strongly recommended to use a uniform size for the user-defined SPH particles.

Input File Usage: **PARTICLE GENERATOR*, NAME=generator name, TYPE=PC3D, MAXIMUM NUMBER OF PARTICLES=number
**PARTICLE GENERATOR INLET*, SURFACE=inlet_surface_name

SPH Particle Outlet

An SPH particle outlet is a tool to remove SPH particles from the flow domain automatically when the SPH particles flow out of the flow domain during the course of the analysis. It is needed when you introduce an SPH outflow boundary condition. You must define an outlet surface associated with the SPH particle outlet. The outlet surface can be thought of as an opening through which the particles flow out of the flow domain. All outlet surfaces associated with an SPH particle outlet must be in one plane (that is, have the same surface normal).



Note: A rigid surface cannot be used as the SPH particle generator outlet surface.

Input File Usage: **PARTICLE OUTLET*, NAME=outlet name, TYPE=SURFACE, SURFACE=outlet_surface

Initial Conditions

Initial Conditions describes all the initial conditions that are available for an explicit dynamic analysis. Initial conditions pertinent to mechanical analyses can be used in a smoothed particle hydrodynamic analysis.

Boundary Conditions

Boundary conditions are defined as described in *Boundary Conditions*. The default interaction between SPH particles and Lagrangian surfaces is based on node-to-surface contact. To improve the SPH solution near the Lagrangian boundaries for flow-type problems, you can define the surface behavior explicitly.

Improving the SPH Boundary Behavior near Lagrangian Boundaries

In the SPH method, particles interact with their neighboring particles through a smoothing kernel function. The smoothing kernel function has a finite supporting domain. To achieve accurate SPH simulation results, there should be enough neighboring particles (50–60) within the supporting domain of the smoothing kernel. However, special considerations are needed near the boundaries. When particles are close to Lagrangian boundaries, part of the supporting domain of the smoothing kernel (that beyond the Lagrangian boundaries) will not be filled with any SPH particles. Therefore, the integration accuracy for those SPH particles close to the Lagrangian boundaries may be negatively affected. In addition, the disturbance of the solution close to the boundaries might eventually lead to inaccurate simulation results for the whole field.

To improve the SPH solution near the Lagrangian boundaries, you can specify surface behavior that is based on the ghost particle method (*Colagrossi and Landrini, 1977*) and define the tangential-type of SPH boundary behavior. When SPH particles are close to Lagrangian boundary surfaces, a virtual plane is formed by taking the average of the adjacent boundary surfaces; based on this virtual plane, ghost particles are created to help improve integration accuracy. The state (pressure, density, velocity, etc.) of the ghost particles is computed from

the real physical particles. With the improved boundary condition, SPH particles on opposite sides of a surface do not interact with each other.

Input File Usage: Use one of the following options to specify the SPH surface behavior:

**SPH SURFACE BEHAVIOR*, TYPE=FREE SLIP (default)
**SPH SURFACE BEHAVIOR*, TYPE=NO SLIP

SPH Inflow Boundary Condition

You can specify velocity or natural boundary conditions on inlet surfaces associated with an SPH particle generator. When you specify a velocity boundary condition on an inlet surface, SPH particles are introduced into the flow domain through the inlet with a user-specified flow speed. The flow direction of the particles is along the normal to the inlet facets. When you specify a natural boundary condition on an inlet surface, SPH particles are introduced into the flow domain whenever the closest layer of SPH particles to the inlet surfaces separates from the surface. The natural inflow boundary condition is usually used with a gravity load.

Input File Usage: Use one of the following options to specify the SPH inflow boundary behavior:

**PARTICLE GENERATOR FLOW*, DEFINITION=SPEED (default)
**PARTICLE GENERATOR FLOW*, DEFINITION=NATURAL

SPH Outflow Boundary Condition

You can specify nonreflecting or pressure boundary conditions on outlet surfaces associated with an SPH particle outlet. When you specify a nonreflecting boundary condition on an outlet surface, the pressure gradient on the outlet surface associated with an SPH particle outlet is zero. When you specify a pressure boundary condition on an outlet surface, the pressure on the outlet surfaces associated with an SPH particle outlet is the user-specified pressure.

Input File Usage: Use one of the following options to specify the SPH outflow boundary behavior:

**PARTICLE OUTLET FLOW*, BOUNDARY CONDITION=NONREFLECT (default)
**PARTICLE OUTLET FLOW*, BOUNDARY CONDITION=PRESSURE

Loads

The loading types available for an explicit dynamic analysis are explained in *About Loads*. Concentrated nodal loads can be applied as usual. Gravity loads are the only distributed loads allowed in a smoothed particle hydrodynamic analysis.

Smoothed Particle Hydrodynamic Domain

The region within which the particle search (finding all neighbors for all particles) is performed is discussed in *Using Section Controls to Define the Particle Tracking Box for DEM and SPH Particles*.

Material Options

Any of the material models in Abaqus/Explicit can be used in a smoothed particle hydrodynamic analysis.

Elements

The smoothed particle hydrodynamic method is implemented via the formulation associated with PC3D elements. These 1-node elements are simply a means of defining particles in space that model a particular body or bodies.

These particle elements utilize existing functionality in Abaqus to reference element-related features such as materials, initial conditions, distributed loads, and visualization.

You define these elements in a similar fashion as you would define point masses. The coordinates of these points lie either on the surface or in the interior of the body being modeled, similar to the nodes of a body meshed with brick elements. For more accurate results, you should strive to space the nodal coordinates of these particles as uniformly as possible in all directions.

An alternative to directly defining PC3D elements is to define conventional continuum finite element types C3D8R, C3D6, or C3D4 and automatically convert them to particle elements at the beginning of the analysis or during the analysis, as discussed in *Finite Element Conversion to SPH Particles*.

The smoothed particle hydrodynamic method implemented in Abaqus/Explicit uses a cubic spline as the interpolation polynomial and is based on the classical smoothed particle hydrodynamic theory as outlined in the references below.

The smoothed particle hydrodynamic method is not implemented for two-dimensional elements. Axisymmetry can be simulated using a wedge-shaped sector and symmetry boundary conditions. There are no hourglass or distortion control forces associated with PC3D elements. These elements do not have faces or edges associated with them.

SPH Kernel Interpolator and Formulations

By default, the smoothed particle hydrodynamic method implemented in Abaqus/Explicit uses a cubic spline as the interpolation polynomial. Alternatively, you can choose a quadratic (*Johnson et al, 1996*) or quintic (*Wendland, 1995*) interpolator.

The implementation is based on the classical smoothed particle hydrodynamic theory as outlined in the references below. You also have the option of using a mean flow correction configuration update, commonly referred to in the literature as the XSPH method (see *Monaghan, 1992*), as well as the corrected kernel of *Randles and Libersky, 1997*, also referred to as the normalized SPH (NSPH) method. In the XSPH method, the mean velocity filtering coefficient is required for modifying the coordinate update for the particles. A zero value for this coefficient leads to the classical SPH method.

You can control these settings as discussed in *Using Section Controls for Smoothed Particle Hydrodynamics (SPH)*.

Computing the Particle Volume

There is currently no capability to automatically compute the volume associated with these particles. Hence, you need to supply a characteristic length that will be used to compute the particle volume, which in turn is used to compute the mass associated with the particle. It is assumed that the nodes are distributed uniformly in space and that each particle is associated with a small cube centered at the particle. When stacked together, these cubes will fill the overall volume of the body with some minor approximation at the free surface of the body. The characteristic length is half the length of the cube side. From a practical perspective, once you have created the nodes, you can use the half-distance between two nodes as the characteristic length. Alternatively, if you know the mass and density of the part, you can compute the volume of the part and divide it by the total number of particles in the part to obtain the volume of the small cube associated with each particle. Half of the cubic root of this small volume is a reasonable characteristic length for this particle set. You can check the mass of individual sets in the model if you request that model definition data be printed to the data (`.dat`) file (see *Model and History Definition Summaries*).

Input File Usage: Use the following options to define a smoothed particle hydrodynamic body:

**ELEMENT*, TYPE=PC3D, ELSET=*particle_body*
element number, *node number*

Repeat the data line as often as necessary.

**SOLID SECTION*, ELSET=*particle_body*, MATERIAL=*material_name*
characteristic length associated with particle volume

Smoothing Length Calculation

Even though particle elements are defined in the model using one node per element, the smoothed particle hydrodynamic method computes contributions for each element based on neighboring particles that are within a sphere of influence. The radius of this sphere of influence is referred to in the literature as the smoothing length. The smoothing length is independent of the characteristic length discussed above and governs the interpolation properties of the method. By default, the smoothing length is computed automatically. As the deformation progresses, particles move with respect to each other and, hence, the neighbors of a given particle can (and typically do) change. Every increment Abaqus/Explicit recomputes this local connectivity internally and computes kinematic quantities (such as normal and shear strains, deformation gradients, etc.) based on contributions from this cloud of particles centered at the particle of interest. Stresses are then computed in a similar fashion as for reduced-integration brick elements, which are in turn used to compute element nodal forces for the particles in the cloud based on the smoothed particle hydrodynamic formulation.

By default, Abaqus/Explicit computes a smoothing length at the beginning of the analysis such that the average number of particles associated with an element is roughly between 50 and 60. The smoothing length can be kept constant or can vary during the analysis. In both cases the average number of particles per element can either decrease or increase during the analysis depending on whether the average behavior in the model is expansive or compressive, respectively.

If the average number of particles per element decreases below a certain minimum number, the deformation gradient might not be calculated accurately due to the limited support from surrounding particles within the sphere of influence. In this case the deformation gradient is kept constant between the previous time increment and the current time increment. By default, the minimum number of particles associated with one element is 5.

You can control most of these settings as discussed in *Using Section Controls for Smoothed Particle Hydrodynamics (SPH)*.

Smoothed Particle Hydrodynamic Domain

By default, all particles are tracked as the analysis progresses. Alternatively, you can define a particle tracking box to define the region within which the particle search (finding all neighbors for all particles) is performed as discussed in *Using Section Controls to Define the Particle Tracking Box for DEM and SPH Particles*.

Constraints

Since the PC3D elements are Lagrangian elements, their nodes can be involved in other features, such as other elements, connectors, or constraints. Since these elements do not have faces or edges, an element-based surface cannot be defined using PC3D elements. Consequently, constraints that require element-based surfaces (such as fasteners) cannot be defined for particles.

Interactions

Bodies modeled with particles can interact with other finite element meshed bodies via contact. The contact interaction is the same as any contact interaction between a node-based surface (associated with the particles) and an element-based or analytical surface. Both general contact and contact pairs can be used. All interaction types and formulations available for contact involving a node-based surface are allowed, including cohesive behavior. Different contact properties can be assigned via the usual options. By default, the particles are not part of the general contact domain similar to other 1-node elements (such as point masses). The default contact thickness for particles is the same value specified as the characteristic length on the section definition, and it is constant through the analysis. Thus, for contact purposes, particles behave as spheres with radii equal to the radius of a sphere inscribed in the small cube associated with the particle volume as described above.

You should not specify a contact thickness of zero for the nodes associated with PC3D elements or contact might not be resolved robustly. The recommended approach is to use the default or specify a reasonable contact thickness.

Interaction between different bodies all modeled with PC3D elements is allowed. However, this interaction is meaningful only in cases when the colliding smoothed particle hydrodynamic bodies are made of the same fluid-like material, such as a water drop falling in a bucket partially filled with water. In solids-related applications, such as modeling a bullet perforating an armor plate, one of the bodies must be modeled using regular finite elements.

Contact interactions cannot be defined between particles and Eulerian regions.

Input File Usage: Use the following options to define contact between a meshed or analytical surface with a particle-based surface:

**CONTACT*
**CONTACT INCLUSIONS*
node-based particle surface, *element-based/analytical_surface*

Output

The element output available for PC3D elements includes all mechanics-related output for continuum elements: stress; strain; energies; and the values of state, field, and user-defined variables. The nodal output includes all output variables generally available in Abaqus/Explicit analyses.

You can visualize particles in Abaqus/CAE by doing the following:

1. Click the **Entity Display** tab.
2. From **View->ODB Display Options**, select **Render element particle edges** .

The display visualization changes depending on the mode or plot type as described in the table below.

Mode/Plot type	Visualization Display
Wireframe	Circular ring for each particle element
Hidden	Combination of a circular ring and disc for each particle element
Filled	Combination of a circular ring and disc for each particle element
Shaded	Combination of a circular ring and disc for each particle element
Contour and Symbol	Combination of a circular rings and colored patches for field output variable values

Limitations

Smoothed particle hydrodynamic analyses are subject to the following limitations:

- They are less accurate in general than Lagrangian finite element analyses when the deformation is not too severe and the elements are not distorted. In higher deformation regimes coupled Eulerian-Lagrangian analyses are also generally more accurate. The smoothed particle dynamic method should be used primarily in cases when the conventional finite element method or the coupled Eulerian-Lagrangian method have reached their inherent limitations or are prohibitively expensive to perform.
- When the material is in a state of tensile stress, the particle motion might become unstable leading to the so-called tensile instability. This instability, which is strictly related to the interpolation technique of the standard smoothed particle dynamic method, is especially noticeable when simulating the stretched state of a solid. As a consequence, particles tend to clump together and show fracture-like behavior.

- Mass distribution in a body defined with particle elements is somewhat different when compared to the mass distribution of the same body defined with continuum elements, such as C3D8R elements. When particle elements are used, the volume of all particles in that body are the same. Consequently, the nodal mass associated with all particles in that body is the same. If the nodes are not placed in a regular cubic arrangement, the mass distribution is somewhat inexact, particularly at the free surface of the body being modeled.
- Surface loads cannot be specified on PC3D elements. However, distributed loads, such as pressure, can be applied to other finite element surfaces that can apply a pressure onto the particle elements via contact interactions.
- Bodies modeled with particles that were not defined using the same section definition do not interact with each other. Hence, you cannot use smoothed particle hydrodynamics to model the mixing of bodies with dissimilar materials.
- Selective subcycling and the contact pair algorithm are not supported.
- Rough friction should not be used on surface interactions defined for SPH particles.
- User-defined PC3D elements and PC3D elements generated from a SPH particle generator should have the same material properties.
- The functionality is not directly supported in Abaqus/CAE. However, you can do the following:
  - You can use the existing functionality in Abaqus/CAE to generate mass elements, write an input file, and then manually edit the input file to convert the mass elements to particles.
  - You can use finite element conversion to SPH particles (see *Boundary Conditions*) at the beginning of an analysis (a time-based conversion criterion at time equal to zero).
  - You can create a mesh using C3D8R elements, write an input file, and then use a script to convert these elements to SPH particles. An example of a Python script to convert solid elements to SPH particles is described in “Generating SPH particle elements from a solid mesh” in the Dassault Systèmes Knowledge Base at *http://support.3ds.com/knowledge-base/*.
- Within a given body (part) defined via one solid section definition, gravity loads and mass scaling cannot be specified selectively on a subset of elements referenced by this definition. Instead, the two features must be applied to all the elements in the element set associated with the solid section definition.

Smoothed particle hydrodynamic computations are distributed across parallel domains in most cases; however, they are all performed by a single domain (with a single processor) for models with any of the following characteristics (which will often dramatically degrade parallel scalability):

- Multiple solid sections for PC3D elements
- Normalized kernel specified as a section control
- Predefined field variable (including temperature) dependence of material properties

Smoothed particle hydrodynamic analyses are subject to the following limitations if multiple CPUs are used:

- Element history output is not supported.
- Dynamic load balancing cannot be activated.
- When contact output is limited to a pair of contact surfaces, SPH particles can only be part of the second surface definition in the contact output request; they cannot be part of the first surface.
- If any SPH particles participate in general contact, all SPH particles must be included in the general contact definition.
- At least 10,000 particles per domain is suggested to achieve good scalability.
- A significant increase in memory usage might be needed if a large number of CPUs are used.

Input File Template

The following example illustrates a smoothed particle hydrodynamic analysis of a bottle filled with fluid being dropped on the floor. The plastic bottle and the floor are modeled with conventional shell elements. The fluid is modeled via smoothed particle hydrodynamics using PC3D elements. The nodal coordinates of the particles are defined such they are all located inside the bottle. Material property definitions are defined as usual for both the fluid and the bottle. Contact interaction is defined between the smoothed particle hydrodynamic particles representing the water (node-based surface) and the interior walls of the bottle and also between the bottle exterior and the floor using element-based surfaces (not shown). Output is requested for stresses (pressure) and density in the fluid.

```
*HEADING
…
*ELEMENT, TYPE=PC3D, ELSET=Fluid_Inside_The_Bottle
Element number, node number
…
*SOLID SECTION, ELSET=Fluid_Inside_The_Bottle, MATERIAL=Water
Element characteristic length associated with particle volume
*MATERIAL, NAME=Water
Material definition for water, such as an EOS material
*ELEMENT, TYPE=S4R, ELSET=Plastic_Bottle
Element definitions for the shells
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
*NSET, NSET=Water_Nodes, ELSET=Fluid_Inside_The_Bottle
*SURFACE, NAME=Water_Surface, TYPE=NODE
Water_Nodes,
*SURFACE, NAME=Bottle_Interior
Plastic_Bottle, SNEG
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
Data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
Water_Surface, Bottle_Interior
**
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=Fluid_Inside_The_Bottle
S, DENSITY
*END STEP
```

References

- Balsara, D., “von Neumann Stability Analysis of Smoothed Particle Hydrodynamics Suggestions for Optimal Algorithms,” *Journal of ComputationalPhysics*, 1995.
- Colagrossi, A., and M. Landrini, “Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics,” *Journal of Computational Physics*, 2003.
- Gingold, R. A., and J. J. Monaghan, “Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars,” *Royal Astronomical Society, Monthly Notices*, vol. 181, pp. 375–389, 1977.
- Johnson, J., R. Stryk, and S. Beissel, “SPH for High Velocity Impact Calculations,” *Computer Methods in Applied Mechanics and Engineering*, 1996.

- Libersky, L. D., and A. G. Petschek, "High Strain Lagrangian Hydrodynamics," *Journal of Computational Physics*, vol. 109, pp. 67–75, 1993.
- Monaghan, J., "Smoothed Particle Hydrodynamics," *Annual Review of Astronomy and Astrophysics*, 1992.
- Monaghan, J., "SPH without a Tensile Instability," *Journal of Computational Physics*, vol. 159, 2000.
- Munjiza, A., and K. R. F. Andrews, "NBS Contact Detection Algorithm for Bodies of Similar Size," *International Journal for Numerical Methods in Engineering*, vol. 43, pp. 131–149, 1998.
- Randles, P. W., and L. D. Libersky, "Recent Improvements in SPH Modeling of Hypervelocity Impact," *International Journal of Impact Engineering*, 1997.
- Swegle, J. W., and S. W. Attaway, "An Analysis of Smoothed Particle Hydrodynamics," *Sandia National Lab Report SAND93–2513*, 1994.
- Wendland, H., "Piecewise Polynomial, Positive Definite and Compactly Supported Radial Functions of Minimal Degree," *Advances in Computational Mathematics*, 1995.

Finite Element Conversion to SPH Particles

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Continuum Particle Elements*
- **CONTACT*
- **INITIAL CONDITIONS*
- **OUTPUT*
- **SECTION CONTROLS*
- **SOLID SECTION*

Overview

You can take advantage of the intrinsic strengths of both Lagrangian finite element and SPH methods when modeling a body. You can define the model with Lagrangian finite elements and convert them to SPH particles either at the beginning of an analysis or after the deformation becomes significant. It is sometimes easier to create the mesh with Lagrangian finite elements, and Lagrangian finite elements are often more accurate for small deformations. SPH methods are well suited for large deformation.

You start by defining a part as usual. You mesh the part with C3D8R, C3D6, or C3D4 reduced-integration elements or a combination of these elements. You then specify that these "parent" elements are to convert to internally generated SPH particles when a user-specified criterion is met. Gravity loads, contact interactions, initial conditions, mass scaling, and output requests associated with the parent elements or nodes of the parent elements will be transferred appropriately to the generated particles on conversion in an intuitive way as explained below. A special formulation is used to ensure the smoothest possible transition between the two modeling methods. The technique can use any of the materials available in Abaqus/Explicit (including user materials).

Activating the Conversion to SPH Particles Functionality

Two conversion techniques are available for converting Lagrangian finite elements to SPH particles: particles can be generated per parent element, or particles can be generated based on a uniform background grid.

The conversion technique of generating particles per parent element is intended to be used when the deformations in the original finite element mesh are significant and elements might distort. Traditionally, in such cases deletion of the soon-to-be distorted Lagrangian elements would be the only choice to allow the analysis to continue. Converting to SPH particles offers an improvement over the element deletion method because the generated particles are able to provide resistance to deformation beyond finite element distortion levels. Consequently, element deletion cannot be used together with element conversion.

The conversion technique of generating particles based on a uniform background grid is intended to generate a uniform distribution of SPH particles at the beginning of the analysis. This is useful in applications where the SPH functionality is the preferred modeling method (such as when modeling fluids) and a uniform distribution of particles is desired because it usually leads to improved accuracy of results.

The element conversion to particles functionality is not active by default.

Generating Particles per Parent Element

You can control the number of particles generated per parent element and choose between one of four criteria to specify when the conversion is to be triggered.

Input File Usage: **SECTION CONTROLS*, ELEMENT CONVERSION=YES, SPH CONVERSION=PER ELEMENT

Abaqus/CAE Usage: Mesh module: **Mesh->Element Type**: **Conversion to particles**: **Yes**

Specifying the Number of Particles to Be Generated

By default, one particle is generated per parent element. You can control the number of particles generated per element by specifying the number of particles to be generated per parent element isoparametric direction. The total number of particles generated per element depends on the element type that is being converted. For example, if you specify 3 particles to be generated per isoparametric direction, on conversion 27 particles would be generated from a C3D8R element, 18 from a C3D6 element, and 10 from a C3D4 element, as illustrated in *Figure 1*. A maximum value of seven particles per direction can be specified. The particles are evenly spaced inside the parent element such that they fill the volume as uniformly as possible. For example, if cubic parent elements are stacked in the user-defined mesh, the particles would be evenly spaced throughout the part.

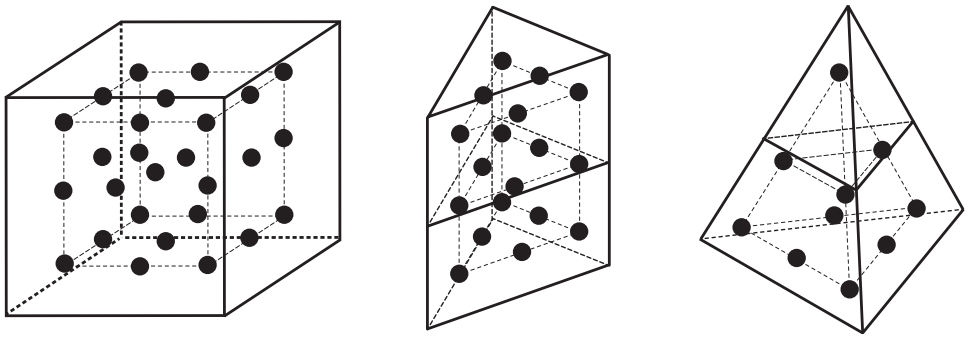


Figure 1: Internally generated particles per parent element illustrated for three particles per isoparametric direction.

You can control the number of particles generated per isoparametric direction as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.

Time-Based Criterion

You can specify the time when the conversion of all the elements in the affected element set is to take place regardless of the deformation levels. This option is intended for applications where the SPH functionality is the preferred modeling method, such as fluid sloshing in a tank or a synthetic bird strike on an aircraft. If the conversion time is specified as zero, the conversion takes place at the beginning of the analysis. For example, fluid sloshing is a good candidate for using a time-based criterion if sloshing is expected to start at the beginning of the analysis. You can specify a later time at which the conversion takes place if extreme deformations do not occur until later in the analysis. A bird strike analysis is a potential candidate as the bird might travel for some time without any deformation prior to hitting the intended target.

You can control the time when the conversion is to occur as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.

Strain-Based Criterion

You can specify the absolute value of the maximum principal strain when the conversion of a given element is to take place. As elements deform, if the absolute value of the maximum principal strain is greater than the specified threshold, the parent elements convert progressively to SPH particles. This option is intended for applications where the finite element method is the preferred modeling method but severe deformations could occur in certain regions. Examples include blast applications and crushing.

You can control the strain-based threshold on which conversion is to occur as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.

Stress-Based Criterion

You can specify the absolute value of the maximum principal stress value at which the conversion of a given element takes place. As elements deform, if the absolute value of the maximum principal stress is greater than the specified threshold, the parent elements will convert progressively to SPH particles. This option is intended for the same candidate applications as those discussed for the strain-based criterion.

You can control the stress-based threshold upon which conversion is to occur as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.

User Subroutine–Based Criterion

The user subroutine–based criterion provides the flexibility of a user subroutine implementation that allows you to implement your own conversion criterion. Element conversion can be controlled during an Abaqus/Explicit analysis through any of the user subroutines that can actively modify state variables associated with a material point, such as *VUSDFLD* and *VUMAT*. You specify the state variable number controlling the element conversion flag. For example, specifying a state variable number of two indicates that the second state variable is the conversion flag in the user subroutine. The conversion state variable should be set to a value of one or zero. A value of one indicates that the element is active, while a value of zero indicates that Abaqus/Explicit should convert the element to particles. Since user subroutines have access via arguments (or in the case of the *VUSDFLD* subroutine via utility routines) to material point state data, the functionality provides a comprehensive means to define the conversion state variable.

Input File Usage: Use the following options to define a user subroutine–based conversion criterion:

**SECTION CONTROLS*, ELEMENT CONVERSION=YES, SPH
CONVERSION=PER ELEMENT
CONVERSION CRITERION=USER
...
**MATERIAL*
**DEPVAR*, CONVERT=*variable number*

Abaqus/CAE Usage: Specifying a user subroutine–based criterion for element conversion is not supported in Abaqus/CAE.

Conversion to Particles Formulation

When using the conversion technique of generating particles per parent element, particles are generated internally at the beginning of the preprocessing phase of the analysis, and they are placed in an inactive or dormant state. The particles are attached to the parent elements in a similar fashion as the nodes of embedded elements are attached (see *Embedded Elements*), and they follow the motion of the parent element nodes in an average sense. The inertial properties of the particles in this inactive state (while the parent finite elements are active) are disregarded automatically to avoid doubling the momentum at a given location. Similar to SPH particles defined directly as PC3D elements, particles generated from parent element sets associated with different section definitions will not interact with each other.

On conversion a number of internally generated particles per parent element are activated, as illustrated for various element types in *Figure 1*. The computational cost of the analysis can increase significantly after conversion takes place if a large number of particles are generated per element since a larger number of active elements needs to be processed. In addition, the computational cost increases because the stable time increment associated with the internally generated particles decreases as the particle density increases.

On conversion the state information (such as stress or equivalent plastic strain) associated with the element being converted is transferred to the generated particles to ensure the smoothest possible transition. The activated particles interact via the SPH formalism with both the previously activated particles and the neighboring inactive particles that are still embedded in active parent elements.

Generating Particles Based on a Uniform Background Grid

This conversion technique allows you to generate a uniform distribution of particles based on a uniform background grid. Using an initially uniform distribution of particles is usually recommended since it can help improve the accuracy of the results.

The origin of the background grid is placed at the center of mass of the element set to be converted. The finite elements in the set are converted to SPH particles at the beginning of the analysis. The particles are generated only at grid line intersections that are inside the volume of a parent element. All generated particles have the same volume and mass. Any parent element that does not contain a grid intersection has no generated particles associated with it.

You can control the spacing and orientation of the uniform background grid as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.

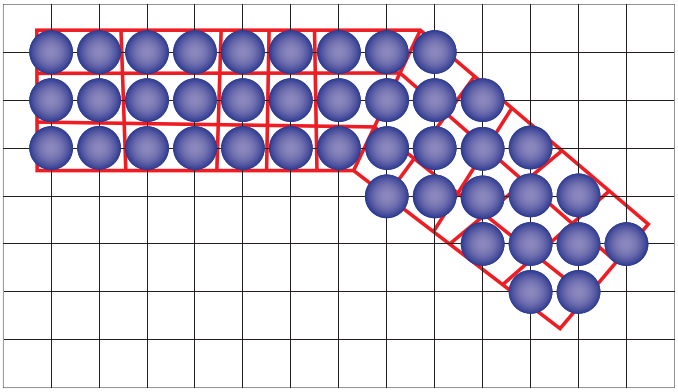
Input File Usage: **SECTION CONTROLS*, ELEMENT CONVERSION=YES, SPH CONVERSION=BACKGROUND GRID (default)

Abaqus/CAE Usage: Generating particles based on a uniform background grid is not supported in Abaqus/CAE.

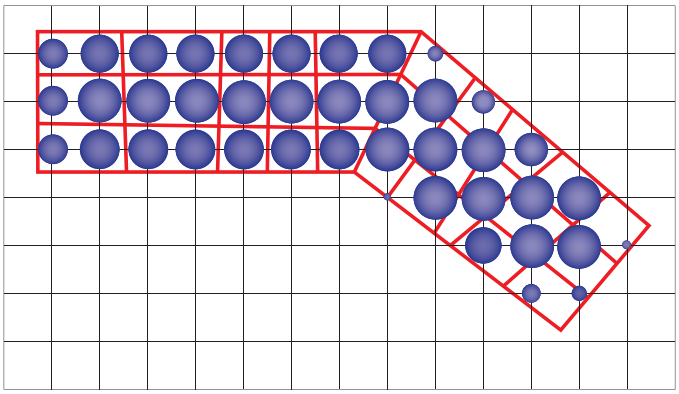
Specifying the Thickness of Generated Particles

The thickness of the particles is primarily used for resolving initial overclosures between the particles and the surfaces in the general contact domain. By default, Abaqus/Explicit determines the particle thickness such that no initial overclosures exist, as illustrated in *Figure 2*(b). You can specify a uniform thickness, although the particle thickness can expand beyond the original surface of the finite element mesh, as illustrated in *Figure 2*(a), which could lead to initial overclosures. The thickness of the particles does not affect the particle's volume and mass.

You can specify the thickness of the generated particles to be either variable or uniform as discussed in *Using Section Controls to Convert Continuum Elements to Particles*.



(a) particles with uniform thickness



(b) particles with variable thickness

Figure 2: Internally generated particles based on a uniform background.

Automatically Generated Sets and Surfaces

Because the particles are generated internally, you do not have the ability to define element sets, node sets, or surfaces associated with these particles. Consequently, a number of sets and surfaces are created internally for convenience. You can visualize these internal sets and surfaces via the usual techniques. *Table 1*, *Table 2*, and *Table 3* describe the internally generated sets and surfaces.

Table 1: Internally generated element sets.

Internally generated element set	Description
ALL_GENERATED_ELEMENTS_SPH	All generated SPH particles in the entire model
ALL_PARENT_ELEMENTS_SPH	All parent elements in the entire model
*UserDefinedElsetName*_SECT_SPH	All generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefined_AElsetName*_SPH	All generated particles associated with the element set *UserDefined_AElsetName*

Table 2: Internally generated node sets.

Internally generated node set	Description
ALL_PARENT_ELEMENT_NODES_E_SPH	All nodes of all parent elements in the entire model
ALL_GENERATED_NODES_SPH	All nodes of all generated particles in the entire model
*UserDefinedElsetName*_SECT_E_SPH	All nodes of generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefined_ANsetName*_SPH	Nodes of generated particles from parent elements touching nodes of the *UserDefined_ANsetName* node set

Table 3: Internally generated surfaces.

Internally generated surfaces	Description
*UserDefinedElsetName*_PARENT_EE_SPH	Element-based surface containing all facets of all elements associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefinedElsetName*_SECT_NE_SPH	Node-based surface with all nodes of all generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefinedSurfaceName*_NS_SPH	Node-based surface containing all nodes of generated particles associated with the elements used in the definition of the *UserDefinedSurfaceName* element-based surface

These sets and surfaces are used by features that are automatically generated internally, such as loads, initial conditions, mass scaling, contact definitions, and output requests. These internally generated features extend the features that you defined for the associated parent sets and surfaces of the internally generated particles. In all cases, the internally generated features preserve the attributes that you defined on the parent sets and surfaces. You cannot define features such as loads, boundary conditions, and initial conditions on the internally generated sets and surfaces directly.

Initial Conditions

Initial conditions (see *Initial Conditions*) cannot be specified directly for the generated particles. However, a subset of the possible initial conditions (stresses, velocity, and rotating velocity) is applied to the generated particles automatically. You specify these initial conditions on the original element or node set you have defined in the model, and they are applied appropriately to the associated generated particles. The initial conditions are applied via the internally created sets described above; hence, you must use an element or node set rather than element or node numbers when applying initial conditions.

Initial stresses specified on parent elements are applied to the generated particles. This feature is leveraged in cases where parent elements convert to particles at the very beginning of the analysis (time zero). All other initial conditions associated with elements are taken into account for the generated particles if the parent elements convert to particles after the first increment in the analysis. The state transfer mechanism described above appropriately transfers the information to particles and, hence, initial conditions are accounted for correctly in the particles.

Boundary Conditions

Boundary conditions (see *Boundary Conditions*) cannot be applied directly to the generated particles. Boundary conditions applied to nodes of the parent elements are not transferred to the generated particles. However, you can use contact interactions to enforce boundary conditions as explained in *Interactions*.

Temperature and field variables specified on node sets that include parent element nodes are extended to the generated particles. Abaqus/Explicit generates corresponding temperature and field variables definitions internally via the internal node sets described in *Automatically Generated Sets and Surfaces*. If all of the nodes of a particular parent element have the same value at a given time, the generated particles would have that same value as well. If different values are specified, no interpolation occurs. Instead, the value of the last definition is used.

Loads

The loading types available for an explicit dynamic analysis are explained in *About Loads*. Concentrated nodal loads cannot be applied to generated particles. Gravity loads specified on the parent elements are the only distributed loads that are transferred upon conversion to the generated particles.

Material Options

Any of the material models in Abaqus/Explicit can be used with the conversion technique.

Elements

When using the conversion technique and C3D8R, C3D6, and/or C3D4 reduced-integration parent elements to define the part, PC3D elements are generated internally at the beginning of the analysis; the parent elements are active, and the PC3D elements are inactive. Upon conversion the active status switches. At no time are a parent element and the associated generated particles both active. By default, the Visualization module automatically displays only the elements that are active at any given time.

Particle mass (and volume) is computed automatically from the mass (volume) of the parent element. All particles associated with a specific parent element will have the same mass (volume). The SPH smoothing length and domain required for the SPH formalism are computed in the same fashion as in the case when you define PC3D elements directly (see *Smoothed Particle Hydrodynamics*).

If mass scaling is defined on element sets containing parent elements, Abaqus/Explicit internally generates mass scaling definitions associated with the corresponding internal element sets described in *Automatically Generated Sets and Surfaces*.

Constraints

Constraints such as couplings or ties cannot be applied directly to the generated particles. However, constraints can be defined on nodes and surfaces associated with the parent element nodes and faces. If such constraints are used to attach parent elements to other Lagrangian bodies or they are used to drive the motion of a part, care must be exercised when the parent element faces involved in such constraints convert to particles. The constraint may be nullified on parent element conversion and, consequently, the connection to other parts (in the case of tie constraints) or to the driving feature (in the case of coupling constraints) would no longer be realized. Hence, in certain cases you may need to place these constraints far enough from the parent elements that can convert for the constraints to be active throughout the analysis.

Element sets that are marked for possible conversion to particles but that are also part of the rigid body definition will never convert because the rigid body constraint is always enforced on the parent elements.

Interactions

Bodies modeled with elements that may convert to particles can interact with other finite element–meshed or analytical bodies via contact. On conversion the internally generated particles may also interact via contact with these bodies but only via the general contact functionality.

By default, if general contact interactions are included in your model, contact inclusions and exclusions involving internal node-based surfaces associated with the internal particles are generated. User-specified contact inclusions and exclusions referencing element-based surfaces that include convertible elements will also be reflected in

internally generated requests. *Table 4* and *Table 5* show all correspondences. The naming convention used for the internally generated surfaces is explained in *Automatically Generated Sets and Surfaces* above.

Table 4: Internally generated contact inclusions.

User-defined contact inclusion	Internally generated contact inclusions
**CONTACT INCLUSIONS*, ALL EXTERIOR	*blank*, *AllUserElsets*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	None
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

Table 5: Internally generated contact exclusions.

User-defined contact exclusion	Internally generated contact exclusions
Always, regardless of user definitions	*UserElemBased*_PARENT_EE_SPH, *UserElemBased*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	None
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

As shown in the second row of *Table 5*, contact between the generated particles and the faces of the associated parent elements is always excluded from the general contact domain. The activated internal particles will interact with the neighboring yet inactive particles still attached to parent elements with exposed faces via the SPH formalism.

The contact interaction for the generated particles is the same as any contact interaction between a node-based surface (associated with the internal particles) and an element-based or analytical surface. All interaction types and formulations available for contact involving a node-based surface are allowed, including cohesive behavior. Different contact properties can be assigned via the usual options. The contact control and property assignment options used for pairs of surfaces that involve parent elements that can convert to particles will be reflected in internally generated assignments for the internal particle-based surfaces. *Table 6* shows the internally generated assignments associated with user-defined requests.

Table 6: Internally generated contact control and property assignments.

User-defined contact inclusion	Internally generated contact inclusions
blank, *blank*	*blank*, *AllUserElsets*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	*UserElemBased*, *UserElemBased*_NS_SPH
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

The generated particles may have different contact thicknesses since they are computed automatically at the beginning of the analysis. If one or two particles per isoparametric direction are requested to be generated upon conversion, all generated particles will have a contact thickness such that they are barely touching the closest face of the parent element. If three or more particles per direction are requested, some of the particles will not be touching the faces of the parent element. For these particles, the contact thickness will be the minimum thickness of all of the particles that are touching the parent element faces on that parent element.

You can specify the contact thickness of the generated particles by using the surface property assignment option for an element-based surface that includes the faces of the parent elements. This modeling choice affects contact interactions on parent elements before they convert.

Output

Output requests associated with parent elements, nodes of parent elements, or contact involving faces of parent elements trigger the creation of output requests associated with the corresponding internally generated particles. For example, if you request element output for an element set that contains parent elements, Abaqus/Explicit automatically creates an additional element output request using the corresponding internal element set containing generated particles, as described in *Automatically Generated Sets and Surfaces*.

A field output request for the STATUS output variable is created automatically for all parent elements and generated particles. The value of the STATUS output variable is toggled automatically between a value of zero and one upon conversion for both parent and generated particles. By default, only the active elements are displayed in the Visualization module. In addition, contour and vector plots are displayed appropriately on the elements that are currently active.

History output requests are also replicated for the generated particles. Since the actual element or node numbers of generated particles are defined internally, you can query the actual number of a particle in the Visualization module before identifying which output curve to display. For example, assume that you requested equivalent plastic strain history output for a small element set containing three C3D8R parent elements and that you requested that two particles per isoparametric direction (eight particles per parent element) are to be generated upon conversion. Before conversion you would have 3 curves to analyze; but after the three elements are converted, there are 24 curves from which to choose. You can query the element number of a particle and then select that curve from the 24 available history curves. Before conversion the curves associated with the particles have a value of zero. Upon conversion there will be a jump to the equivalent plastic strain value at the current time.

Limitations

Analyses involving finite element conversion to SPH particles are subject to the following limitations:

- Only reduced-integration continuum elements C3D8R, C3D6, and C3D4 are available for conversion.
- Surface loads specified on the faces of parent elements that convert during the analysis are not applied after conversion to particles. However, distributed loads, such as pressure, can be applied to other finite element surfaces that do not convert (for example, on a piston surface) that can apply a pressure onto the particle elements (for example, the fluid pushed by the piston) via contact interactions.
- Bodies modeled with elements that may convert to particles that were not defined using the same section definition will not interact with each other between the converted portions of the bodies. For example, body A and body B allow elements to convert to particles, but these elements are associated with different section definitions. After conversion, the particles will not interact.
- Within a given body (part) defined via one solid section definition, gravity loads and mass scaling cannot be specified selectively on a subset of elements referenced by this definition. Instead, the two features must be applied to all the elements in the element set associated with the solid section definition.
- Progressive conversion of finite elements into SPH particles during an analysis (based on strain, stress, or user-defined criterion) should be used only in applications that are inertia dominated and for which at any point during the analysis the strain energy is a small percentage of the total energy in the system. Specifically, progressive conversion should be used only in applications involving severe deformations, such as hypervelocity impact, blast, and crushing.
- Two types of internally generated particles cannot be used in the same analysis; therefore, you cannot use element conversion (which internally generates SPH particles) with the SPH particle generator and SPH particle outlet described in *Smoothed Particle Hydrodynamics*.

Input File Template

The following example illustrates a smoothed particle hydrodynamic analysis of a bottle filled with fluid being dropped on the floor using the conversion technique. The plastic bottle and the floor are modeled with conventional shell elements. The fluid is modeled with C3D4 elements that will convert to two particles per isoparametric direction (four particles per element) at the beginning of the analysis based on a time-based criterion. Material property definitions are defined as usual for both the fluid and the bottle. Contact interaction is defined using the default options. Output is requested for stresses (pressure) and density in the fluid.

```
*HEADING
…
*ELEMENT, TYPE=C3D4, ELSET=Fluid_Inside_The_Bottle
…
*SOLID SECTION, ELSET=Fluid_Inside_The_Bottle, MATERIAL=Water,
CONTROLS=Time_Based_Conversion
*SECTION CONTROLS, ELEMENT CONVERSION=YES,
CONVERSION CRITERION=TIME, NAME=Time_Based_Conversion
First data line
Second data line
Third data line
2, 0.0
*MATERIAL, NAME=Water
Material definition for water, such as an EOS material
*ELEMENT, TYPE=S4R, ELSET=Plastic_Bottle
Element definitions for the shells
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
Data lines to define gravity load
**
*CONTACT
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=Fluid_Inside_The_Bottle
S, DENSITY
*END STEP
```

Particle Generator

Products: Abaqus/Explicit Abaqus/Viewer

References:

- *Discrete Element Method*
- *Discrete Particle Elements*
- **PARTICLE GENERATOR*
- **PARTICLE GENERATOR INLET*
- **PARTICLE GENERATOR MIXTURE*
- **PARTICLE GENERATOR FLOW*
- **PROBABILITY DENSITY FUNCTION*

Overview

The particle generator:

- is a tool for automatically introducing discrete particles (PD3D) or SPH particles (PC3D) into the problem domain during the course of the analysis;
- can be used during an initial step to prepare a discrete element method (DEM) model prior to the actual analysis; and
- can be used to continuously introduce discrete particles or SPH particles into the model while the analysis is in progress.

Introduction

The particle generator capability enables a modeling approach in which the discrete element method considers particles only after entering a region of interest. This approach can simplify model set-up and improve computational efficiency. Characteristics of this capability include:

- particles are generated at an inlet that can vary in shape over time,
- generated particles can have random sizes based on a user-specified probability density function,
- multiple species of particles can be generated simultaneously by a single generator, and
- the net mass flow rate and particle speeds at the inlet can be specified with time dependence.

Applications

How you use the particle generator will depend on the process being analyzed. Two broad usage categories are as follows:

Model creation

In many applications the particle generator can be used to fill part of a domain with particles to establish initial conditions for subsequent studies. *Figure 1* shows an example where two particle generators are filling a drum mixer with two different batches of particles prior to performing a mixing simulation. The sequence of images show the start and end of the particle generation phase.

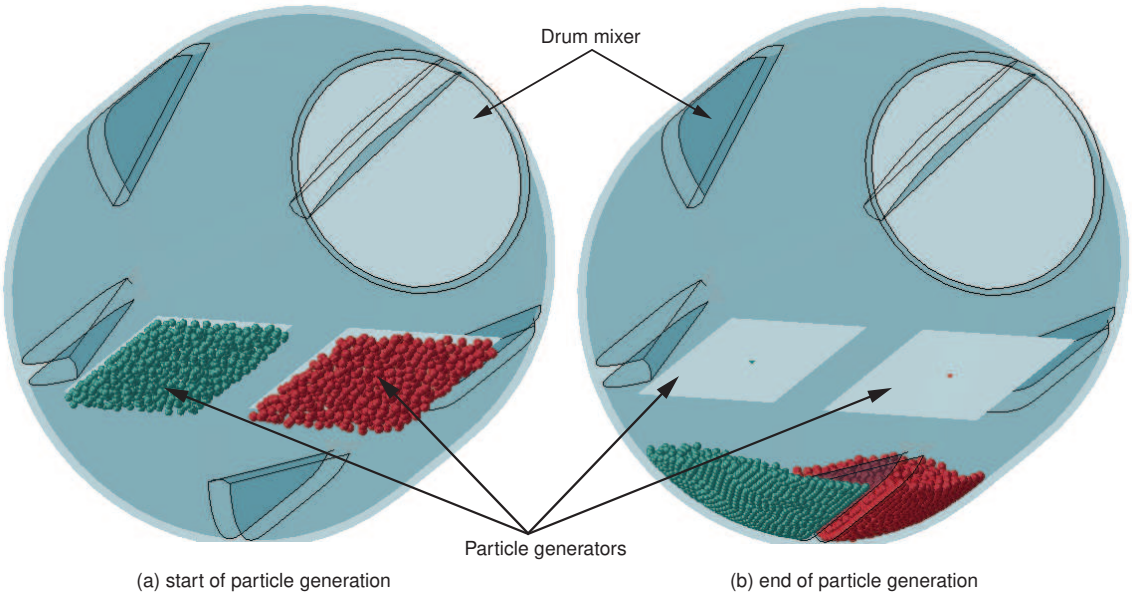


Figure 1: Model creation using particle generator.

Continuous generation

In another type of application a particle generator can be used to inject a continuous stream of particles into the domain while an event of interest involving particles that have been introduced is underway. *Figure 2* shows the use of a particle generator in a sorting analysis. In this application a particle generator is continuously feeding random sized particles to a screen sorter assembly while the analysis is in progress.

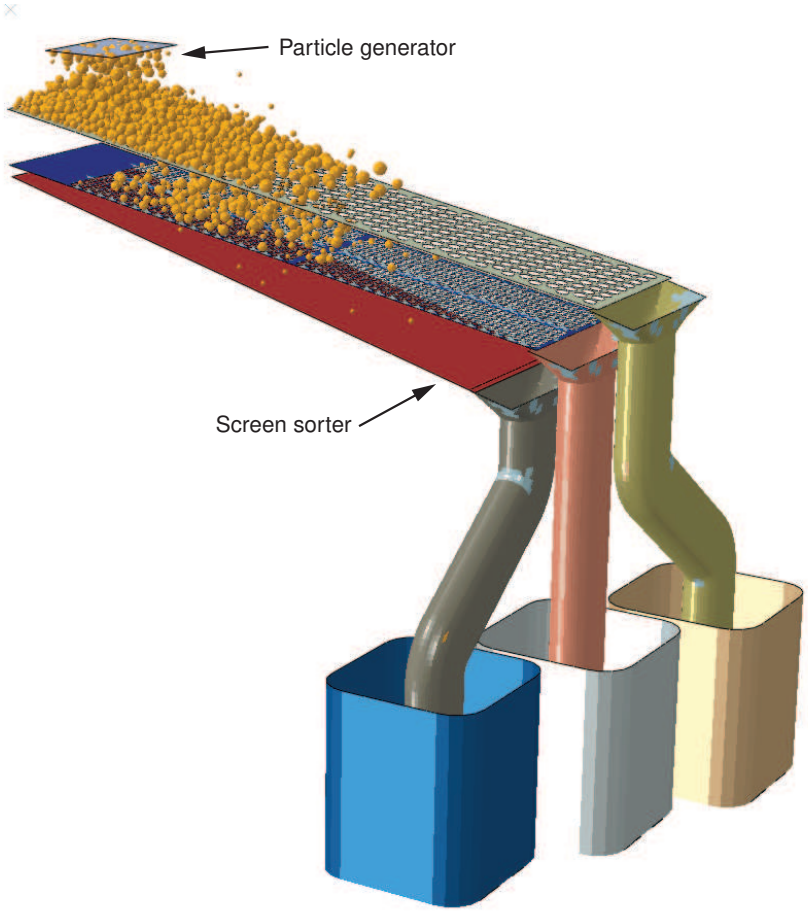


Figure 2: Continuous particle generation example.

Defining a Particle Generator

To define a particle generator, you specify a name for the generator, the type of particle to be generated, and the maximum number of particles that might be generated during the analysis. You must define an inlet surface, the list of particle species that will be generated, and the size distribution for each species.

In many processes the number and speed of the particles being introduced into the domain are not constant during the analysis. You must define the time-dependent mass flow rate per unit inlet area and the particle speed at the generator inlet to control the rate of injection of particles of each species through the inlet. These specifications allow the possibility of turning the particle generator on or off during the analysis.

Input File Usage: **PARTICLE GENERATOR*, NAME=*generator_name*, TYPE=PD3D,
MAXIMUM NUMBER OF PARTICLES=*number*

Inlet Surface

The inlet surface can be thought of as an opening through which the particles are injected into the domain. It is important that no other model feature is positioned in front of the generator inlet to obstruct the flow of particles. An element-based surface must be defined for the inlet surface. Using surface elements is appropriate in most applications. Care should be taken with respect to the positive surface normal direction at the inlet because the

initial particle velocity will be in this direction. A complex inlet geometry as shown in *Figure 3* might have several facets. In this example the normal direction to the inlet surface is out of the page.

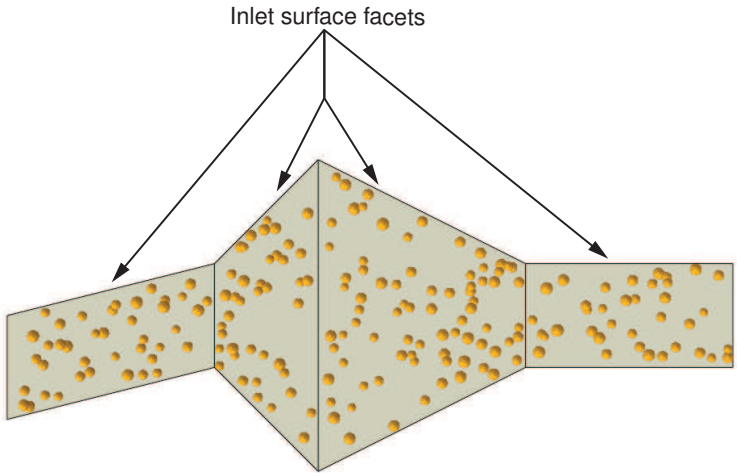


Figure 3: Complex inlet geometry with multiple facets.

The size and shape of the inlet surface can be modified during an analysis by adjusting the nodal positions of the inlet surface. The inlet surface can expand, collapse, and translate as well as undergo rigid body motions. *Figure 4* shows the inlet surface of four different generators being subjected to different deformation and motion.

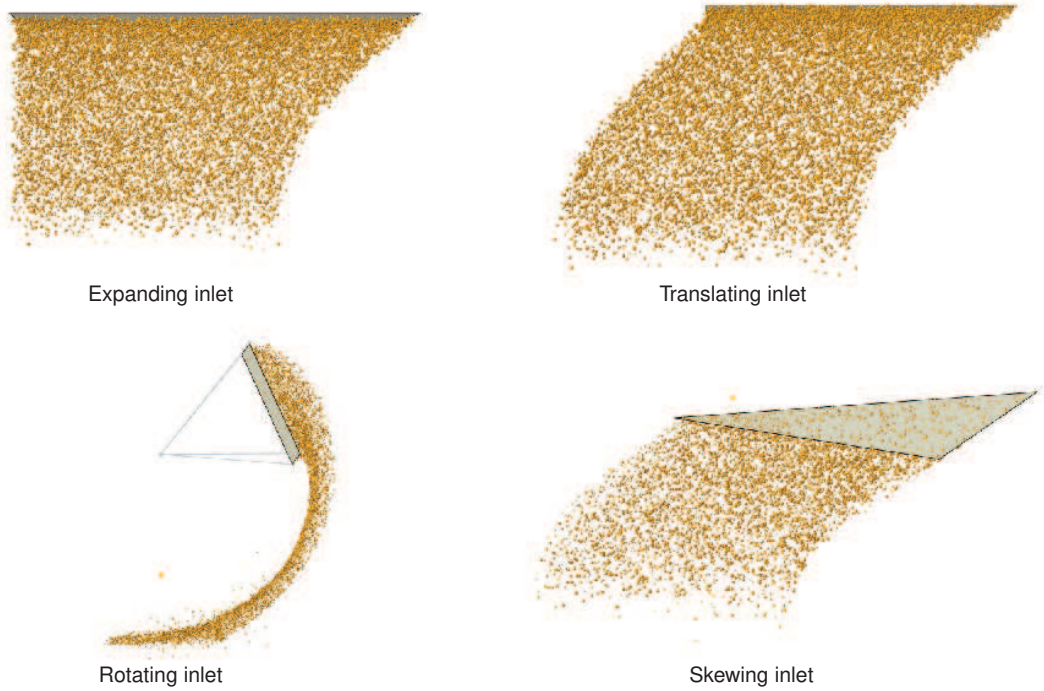


Figure 4: Expanding, translating, rotating, and skewing an inlet surface.

Input File Usage: **PARTICLE GENERATOR INLET*, SURFACE=*inlet_surface_name*

Particle Species

In many applications several different types of particles interact with each other. For example, sand, cement, and aggregate particles are mixed together to prepare a batch of concrete. A particle generator can be used to generate one or more species of particles as shown in *Figure 5*, where the two colors represent the two species of particles that have been simultaneously generated by a particle generator. An element set name is associated with each species in the mixture. This is analogous to a regular DEM analysis where the particles are grouped into element sets.

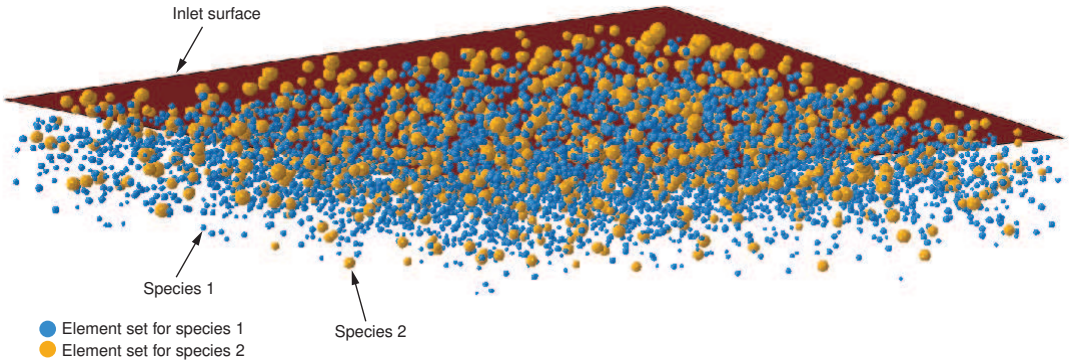


Figure 5: Generation of two species of particles.

Input File Usage: **PARTICLE GENERATOR MIXTURE*
element set for species 1, *element set for species 2*, *etc.*

Particle Size Distribution

Statistical distributions are commonly used to specify the particle sizes. Statistical distributions are defined by analytical or user-specified piecewise linear tabular probability density functions (PDF). The particle generator supports the following different types of probability density functions: uniform, normal, log-normal, and piecewise linear. The normal and log-normal PDFs have well-known analytical forms that depend on the mean and standard deviation of the random deviate. In some cases the particle size distribution might be obtained from sieving particle samples in the field. These data are usually in the form of a table of size ranges and the percentages of particles that fall within those size ranges (a histogram). The piecewise linear PDF can be used for such data. See *Probability Density Function* for further details on the usage of probability density functions.

Mass Flow Rate

The rate of generation of particles is governed by the specified mass flow rate per unit area at the inlet, the area of the inlet, and the ability of the generator to insert a particle of a given size. The time varying mass flow rate per unit inlet area, $\dot{M}_s(t)$, has the units $\frac{mass}{time*length^2}$. For a multiple species particle generator the total mass flow rate per unit inlet area is the sum of the mass flow rates per unit inlet area of the individual species:

$$\left(\dot{M}_s(t) = \dot{M}_s^{\,1}(t) + \dot{M}_s^{\,2}(t) + \ldots\ldots + \dot{M}_s^{\,n}(t)\right).$$

The time varying mass flow rate per unit inlet area for each species must be provided in the form of an amplitude table (*Amplitude Curves*).

Input File Usage: **PARTICLE GENERATOR FLOW*, GENERATOR=*generator_name*
, mass flow rate amplitude for species 1, *mass flow rate amplitude for species 2*, *etc.*

Particle Speed

For a given particle generator, particles of all species are introduced into the domain through the inlet with a specified flow speed. The flow direction of the particles is along the normal to the inlet facet. Particles generated from different facets in the inlet surface will all have the same flow speed; however, depending on the facet normal direction, they may have different velocities. The time varying particle flow speed must be provided in the form of an amplitude table (*Amplitude Curves*).

Input File Usage: **PARTICLE GENERATOR FLOW*, GENERATOR=*generator_name*
flow speed amplitude curve,

Local Coordinate System for Inlet Faces

The current coordinates of the nodes of each facet in the inlet surface are used to define a plane (a generation plane) where particles will be generated. *Figure 6* shows the local directions of the generation plane. The first local direction, $\vec{t_1}$, is defined by the longest facet edge, while $\vec{t_3}$, the third local direction, is normal to the facet. The normal to the facet is obtained from the cross-product of the diagonal vectors (the vector joining nodes 2 and 4 and the vector joining nodes 1 and 3; see *Figure 6* for local node numbers). The second in-plane direction is $\vec{t_2} = \vec{t_3} \times \vec{t_1}$. The plane of generation is positioned R_{min} in front of the inlet facet, where R_{min} is the smallest particle radius for the generator.

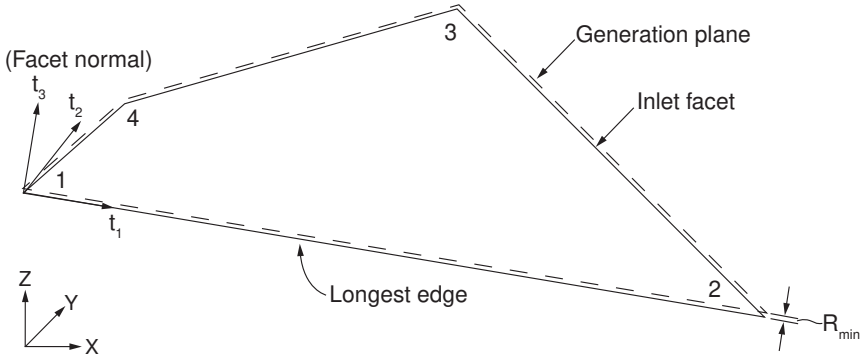


Figure 6: Particle generation plane.

The in-plane particle coordinates of a particle are generated randomly according to a uniform random distribution. The generated in-plane coordinates are acceptable if the particle can be placed at that location ensuring that it does not overlap with previously generated particles and also does not intersect the boundaries defined by the facet edges.

Assigning Properties to Particles

Particles of different species will have different size distribution, density, and contact properties. To assign a density and size distribution for a particle species, the corresponding element set name is specified on a section definition for the PD3D elements. The element set name is also used to define a surface that is referred to under the contact specifications. This way of assigning particle properties and contact properties for generated particles is no different from what is done for user-defined sets of discrete particles. See *Discrete Element Method* for details on how to assign properties to PD3D elements and how to set up contact definitions for a DEM analysis.

Particle Size Rejection

Abaqus/Explicit discards particle sizes that are outside the specified minimum and maximum range for the PDF. For normal and log-normal type PDFs this leads to a truncated distribution, as explained in *Probability Density Function*. Keep this in mind when comparing the generated particle size distribution to the target PDF.

A generated particle size is discarded if it is too large to fit through the current inlet geometry. It is possible that the generator might fail to place a particle in the randomly generated location in the generation plane if there is not sufficient room to accommodate the newly generated particle size. If this happens, the generator will generate another random location and attempt to place the particle there. By default, 15 such attempts will be made, after which the particle size is discarded and another particle size is randomly generated and the process is repeated. You can modify the default number of attempts to place a generated particle size.

If particle sizes are rejected because of either of the above reasons, the generated size distribution cannot be guaranteed to match the specified size distribution. *Figure 7* shows a comparison of generated and specified piecewise linear particle size distributions for a specific problem where 14608 particles were generated at a constant mass flow rate and with constant particle speed through an inlet of fixed geometry. The overall agreement of the generated particle size distribution is good even though the generator was unable to insert some particles into the domain.

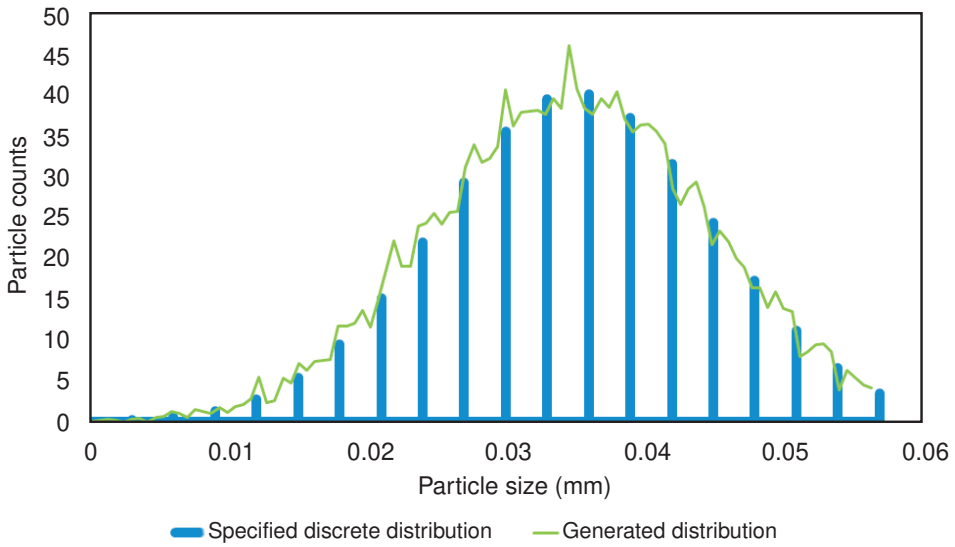


Figure 7: Comparison of specified and generated discrete particle size distributions.

Input File Usage: **PARTICLE GENERATOR*, MAX INSERTION ATTEMPTS=*number*

Mass Balance for Continuous Generation

At any increment, k, corresponding to the end of increment time, t_k, and time increment size, Δt_k, the specified mass of particles of species, i, to be generated is

$$m_{specified_k}^{\ i} = \dot{M}_s^{\ i}(t_k)\, A(t_k)\, \Delta t_k,$$

where $A(t_k)$ is the current inlet facet area. Due to the discrete nature of the time increment and particle sizes, the particle generator will not be able to match the specified mass exactly for the species for the given increment. The particle generator, therefore, maintains a mass balance for each species. For a given species a negative mass balance indicates that for that species the mass of particles generated was higher than what was required. The

excess mass balance for species, i, from the previous increment $m_{balancek-1}^{i}$ is taken into account while computing the target mass to be generated in the current increment:

$$m_{targetk}^{i} = m_{specifiedk}^{i} + m_{balancek-1}^{i}.$$

The mass balance for species i for the current increment is

$$m_{balancek}^{i} = m_{targetk}^{i} - m_{actual}^{i},$$

where m_{actual}^{i} is the actual mass of the particles generated in increment k. The particle generator tries to match the specified mass flow rate per unit inlet area as closely as possible. If the target mass for a species is negative because of a high negative mass balance, no particles will be generated for that species until the target mass becomes positive again. *Figure 8* shows a comparison of generated and specified mass flow rate for a specific problem where the specified mass flow rate per unit inlet area does not vary with time and the inlet geometry is fixed.

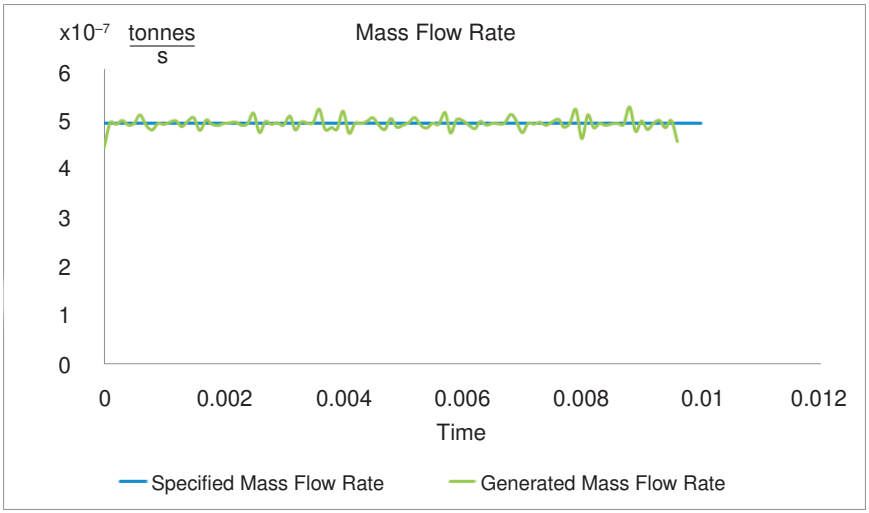


Figure 8: Comparison of specified and generated mass flow rate.

Inlet Blocking

The flow speed, $Q(t)$, and mass flow rate per unit inlet area, $\dot{M}_s(t)$, are related as follows:

$$\dot{M}_s(t) = Q(t)\,\rho\epsilon(t),$$

where ρ is the particle density and $\epsilon(t)$ is the solid fraction. The solid fraction is the ratio of the volume of particles to the volume of the region they occupy. This relation suggests that the packing of particles ahead of the inlet increases for a given mass flow rate per unit inlet area as the flow speed is reduced. If this happens, the particle generator will take many more attempts to insert a particle. Therefore, a meaningful specification of flow speed for a given mass flow rate per unit inlet area is crucial for the efficiency of the particle generator. It is also important that no other model feature is positioned in front of the generator inlet to obstruct the flow of particles.

Automatic Halting of Particle Generator

A particle generator will permanently halt after generating the maximum number of particles. Since it might be difficult to make an accurate estimation of the total number of particles that need to be generated during an

analysis, overestimating the maximum number of particles will ensure that a particle generator will not run out of particles. However, grossly overestimating the maximum number of particles will lead to larger memory requirements for the analysis.

The particle generator will temporarily halt for an increment if the region near the inlet has a high volume fraction of particles. This helps avoid performance degradation due to an excessive number of failed attempts to insert a new particle when the inlet is becoming clogged with existing particles. The default threshold for a temporary halt to the generator is a particle volume fraction of 0.4 over a volume that extends $\mathbf{2.0}R_{max}$ in front of the inlet surface, where R_{max} is the maximum particle radius for the generator. You can modify the default value of the solid fraction used for halting the generator.

It is possible that the current inlet geometry and the particle size distribution is such that the particle generator repeatedly exhausts the 15 attempts to insert a particle of a given size. If 10 such cycles of 15 attempts each have been unsuccessful in a given increment, the generator will temporarily halt generation. You can modify the default value of 10 cycles prior to halting particle generation.

Input File Usage: **PARTICLE GENERATOR*, SOLID FRACTION=*threshold solid fraction*,
MAX ATTEMPTS PER INCREMENT=*number of insertion attempts cycles*

Initial Conditions

Initial conditions should not be applied to nodes of particle species.

Boundary Conditions

Boundary conditions can be applied to the nodes of the inlet facet to expand, shrink, or move the inlet facets of a generator.

Loads

The element set name for a species can be used to apply loads such as gravity or body loads.

Elements

The particle generator generates PD3D and PC3D elements.

Constraints

Constraints such as multi-point constraint type BEAM can be applied to the nodes of inlet facets to facilitate the application of rigid body motion to the inlet facet of a generator.

Interactions

Contact interactions between particle species are specified similar to user-defined PD3D elements. You must not include the inlet facets in the contact domain.

Output

Output is available for the element sets and node sets generated by the particle generator.

Limitations

The following limitations apply to analyses involving the particle generator:

- Only PD3D and PC3D elements can be generated.
- It is not possible to generate clusters of PD3D elements.
- Although supported in Abaqus/Viewer, the functionality is not supported in Abaqus/CAE.

Input File Template

The following example illustrates a particle generator with two particle species:

```
*HEADING
…
*PARTICLE GENERATOR, NAME=generator_name, TYPE=PD3D,
MAXIMUM NUMBER OF PARTICLES=number, SOLID FRACTION=number,
MAX INSERTION ATTEMPTS=number, MAX ATTEMPTS PER INCREMENT=number
**
*PARTICLE GENERATOR INLET, SURFACE=inlet_surf
**
*PARTICLE GENERATOR MIXTURE
gen_SET1, gen_SET2
**
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET1, TYPE=NORMAL
Data line to define PDF
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET2, TYPE=LOGNORMAL
Data line to define PDF
**
*DISCRETE SECTION, ELSET=gen_SET1
PDF_gen_SET1
*DISCRETE SECTION, ELSET=gen_SET2
PDF_gen_SET2
…
*SURFACE, TYPE=ELEMENT, NAME=inlet_surf
element set name
*SURFACE, TYPE=ELEMENT, NAME=PG_SURF1
gen_SET1
*SURFACE, TYPE=ELEMENT, NAME=PG_SURF2
gen_SET2
**
*SURFACE INTERACTION,NAME=prop_gen_SET1
*SURFACE INTERACTION,NAME=prop_gen_SET2
*SURFACE INTERACTION,NAME=prop_gen_SET1_and_gen_SET2
**
*AMPLITUDE, name=flow_speed
*AMPLITUDE, name=mass_flow_rate_gen_SET1
*AMPLITUDE, name=mass_flow_rate_gen_SET2
**
*STEP
*DYNAMIC, EXPLICIT
**
*PARTICLE GENERATOR FLOW, GENERATOR=generator_name
flow_speed, mass_flow_rate_gen_SET1, mass_flow_rate_gen_SET2
**
```

```
*DLOAD
Data lines to define gravity load for gen_SET1
Data lines to define gravity load for gen_SET2
**
*CONTACT
*CONTACT INCLUSIONS
PG_SURF1, PG_SURF1
PG_SURF2, PG_SURF2
PG_SURF1, PG_SURF2
*CONTACT PROPERTY ASSIGNMENT
PG_SURF1, PG_SURF1, prop_gen_SET1
PG_SURF2, PG_SURF2, prop_gen_SET2
PG_SURF1, PG_SURF2, prop_gen_SET1_and_gen_SET2
**
*END STEP
```

Lumped Kinetic Molecular Method

Products: Abaqus/Explicit

References:

- *Particle Generator*
- *Discrete Particle Elements*
- **PARTICLE GENERATOR*

Overview

The lumped kinetic molecular method (LKM) is a particle-based method for modeling gases.
The lumped kinetic molecular method:

- can be used to simulate airbag deployment;
- uses PD3D elements to model lumped gas molecules; and
- uses a particle generator to model an airbag inflator.

Introduction

The lumped kinetic molecular method (LKM) is a particle method that approximates the macroscopic behavior of a gas. It is based on the kinetic theory of gases that assumes that all gases are composed of an enormous number of extremely small molecules in a constant state of random motion. For example, only four grams of helium contain 6.023×10^{23} molecules with a Van der Waals radius of 140×10^{-9} m. The presence of such a large number of molecules allows for the motion of the gas molecules to be treated in a statistical manner. The average behavior of the molecules determines the macroscopic gas behavior. Because numerical modeling of every gas molecule is impractical, the LKM method reduces the size of the problem by lumping many gas molecules into a single gas particle (preserving the macroscopic gas behavior). In the LKM method we solve for the motion of gas particles.

According to the kinetic theory of gases, the Maxwell-Boltzmann distribution describes the speed of the gas molecules at a given temperature. *Figure 1* shows the Maxwell-Boltzmann speed distribution for helium atoms at a temperature of 300 Kelvin. The molecules of gas collide elastically with each other, as well as with the walls of the container. The pressure on the container wall is the result of collisions with the wall. The root mean square (RMS) velocity of the gas particles is given by $V_{rms} = \sqrt{\frac{3KT}{m}}$, where m is the mass of the gas molecule, and T is the absolute temperature of the gas. This link between speed and temperature allows the solution of the gas flow problem in terms of its molecular velocities as a fundamental problem variable.

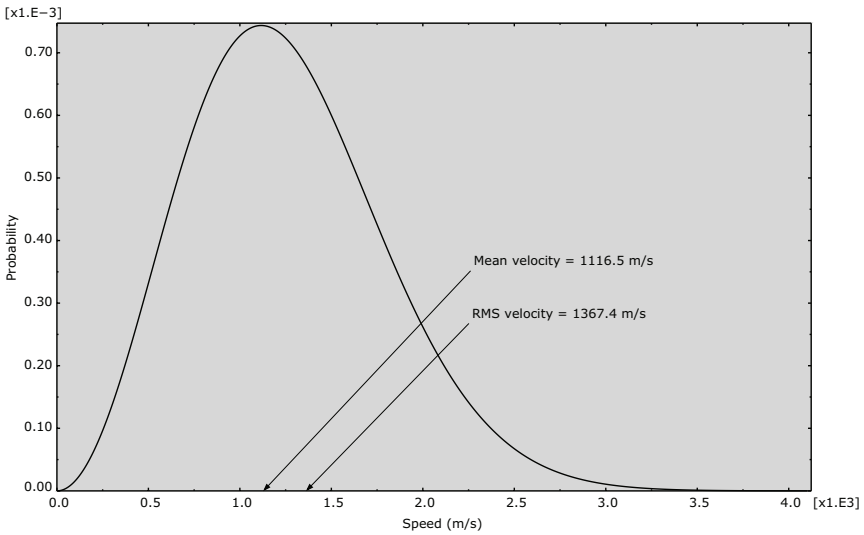


Figure 1: Maxwell-Boltzmann speed function.

The LKM method is based on the following assumptions:

- Lumped particles are rigid spherical particles that collide elastically.
- Lumped particles obey the Maxwell-Boltzmann speed distribution.
- Lumped particles modeling a monatomic gas have only translational energy.
- Lumped particles modeling a polyatomic gas have both translational and rotational energy.
- The temperature of the gas is low enough to ignore vibrational energy.
- No attractive or repulsive forces exist between particles.
- The pressure exerted by the gas on a structure is the combined result of particle collisions over time on the surface.

The accuracy of the prediction of the LKM method depends on the size and the number of lumped molecules. Using too few particles or using particles that are too large can lead to inaccurate solutions. While a large number of particles in general increases the accuracy of the solution, it also increases the computational cost. The optimal number of particles that gives an accurate solution at an acceptable computational cost depends on the amount of gas, the cavity size, and the discretization of the cavity surface mesh. For a small-sized problem, 100,000 gas particles might be sufficient, while for a larger problem, 400,000 gas particles might be required. A general guideline is that the number of particles should be large enough such that the ratio of the average facet mass to the gas particle mass is above 30. This ratio is printed in the status (`.sta`) file. You can use this ratio to check if the number of particles is sufficient for the analysis.

Applications

The LKM method can be used to simulate the deployment of airbags. Such types of analyses are commonly performed as part of the occupant safety assessment of automobiles. Airbags are safety devices that minimize injury to occupants during a vehicle crash. During a crash an airbag undergoes rapid inflation followed by deflation, which cushions the impact between an occupant and the interior of the vehicle. There are various types of airbags, such as the driver airbag that is placed in the steering wheel, the side curtain airbag that is placed in the door panel, and the side torso airbag that is placed in the seat. Most airbags consist of a bag made out of a flexible fabric, an inflator device, an electronic controller, and sensors. Different types of inflators are used for the different types of airbags. Most inflators consist of a cylindrical metal housing that contains the propellant and an igniter. There are orifices in the housing of the inflator through which hot gases escape when the inflator is fired.

The airbag can be tightly folded or wrapped around the inflator to fit the assembly in a confined space such as the steering wheel. In the event of a crash, the controller unit evaluates the signals from the sensors and fires the igniter, setting off a controlled explosion in the inflator. The rapidly expanding gases exit through the orifices in the inflator housing to deploy the airbag. The deployment duration depends on the size and type of airbag and is usually between 20-30 milliseconds. Finally, as the airbag gets squeezed between the occupant and the vehicle, the gases escape through vents in the airbag causing the airbag to deflate.

Figure 2 shows a sectional view of a partially deployed curtain airbag in contact with a rigid hemispherical head form. A close-up view of a tiny volume of the gas in the figure shows the randomly moving gas particles. In the simulation the inflator introduces gas particles inside the airbag, imparting each particle a random speed based on the Maxwell-Boltzmann speed function and a random direction for its velocity. The impact of the fast moving gas particle against the facets of the airbag results in the transfer of energy between the gas and the airbag. The speed of a gas particle decreases as it collides with a facet that is moving away from it, and the speed increases when the gas particle collides with a facet that is moving towards it. All collisions are elastic. The inflator continues to introduce more gas particles into the airbag, pushing it open. As the speed of more gas particles decrease after collision with the expanding airbag, the root mean square (RMS) velocity of the gas particles decreases. This decrease in the average velocity of the lumped molecules is equivalent to the cooling of the injected gas.

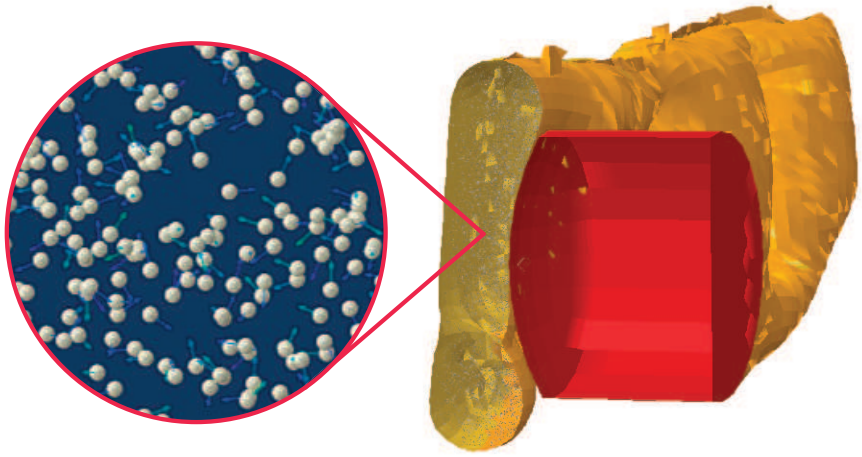


Figure 2: Partially deployed airbag.

Model Setup

A fluid cavity must be associated with the airbag interior surface. The fluid cavity is used only to calculate a cavity volume. The cavity volume is used to compute the output average pressure in the cavity. The fluid cavity does not contribute any pressure loading. The pressure loading is due only to particle impacts.

For the LKM method, you must specify the universal gas constant and the Ludwig Boltzmann constant.

Input File Usage: Use the following options to define the constants and the fluid cavity:

**PHYSICAL CONSTANTS*, UNIVERSAL GAS CONSTANT=*R*, BOLTZMANN=*K*
…
**FLUID CAVITY*, REF NODE=*flucavrefnode*, SURFACE=*airbag_surface*, ADIABATIC,
BEHAVIOR=*air_gas*, AMBIENT PRESSURE=*ambient pressure value*

Defining an Inflator

An inflator is a device that injects gas into an airbag during deployment. The particle generator works as an inflator for the LKM method. During the analysis, the particle generator injects gas particles at specific locations inside the airbag. The particle generator must be associated with the gas behavior and with the mass flow rate and the temperature history of the inflator to determine the number of particles to add per time increment and their velocities. The fluid cavity associated with the airbag must also be associated with the particle generator. See *Particle Generator* for further details.

LKM particles of a given gas species have the same size. The size is determined automatically from the estimated maximum airbag volume and the ratio of the assumed mean free path to the size of the lumped molecule. The actual mean free path is the average distance lumped particles travel between collisions, and it is an unknown quantity. You must specify the maximum airbag volume in the particle generator definition. A reasonable estimate of the maximum airbag volume results in a good balance between accuracy and performance.

The ratio of the assumed mean free path to the size of a lumped molecule is used only for determining the size of the gas particles. The default value of this ratio is 500. The default value works well for most airbag models. To reduce the automatically calculated particle size, you can increase the value of this ratio. The actual ratio of the mean free path to the particle size is an outcome of the solution and depends on the particle concentration during the analysis.

Defining Inflator Geometry

Inflators come in different shapes and sizes depending on the airbag in which they are used. Typically, the gas is injected through orifices that are located around the inflator housing. *Figure 3* shows a schematic diagram of a side-firing inflator device with orifices arranged around the side of the inflator housing. The particle generator that works as the inflator in the LKM method only requires the location and orientation of the orifices of the inflator. A single planar facet can be used to approximate an orifice. *Figure 4* shows the inlet facets of the corresponding particle generator model. The center of each facet coincides with each of the corresponding nine inflator orifice centers. The outward normal shown on each of the facets indicates the direction in which the gas particles are generated. Gas particles are generated in random directions in the front half plane of each facet of the inlet surface. The facets together form the inlet surface of the particle generator. It is recommended that you use surface elements to define the inlet facets. The particle generator uses the inlet facets merely as geometrical entities through which particles are injected into the problem domain. The gas particles do not have any contact interaction with the inlet facet. Therefore, it is important that you ensure that contact is not defined between the inlet surface and the gas particles.

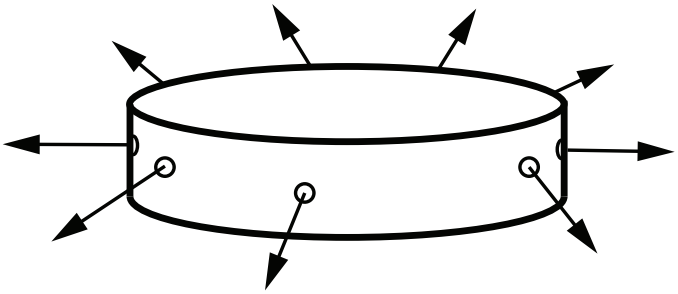


Figure 3: Inflator.

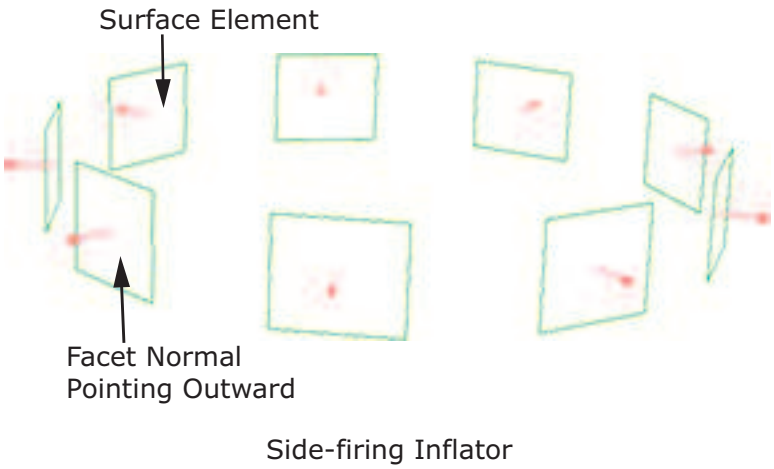


Figure 4: Inflator modeled with particle generator.

The inlet facet should be many times larger than the actual particle diameter. This ensures that the particle generator is able to inject a large number of particles without blocking the inlet facets. The approximation of the orifice geometry and size with a single planar facet does not have a significant influence on the accuracy of the solution. The location of the inlet facets inside the airbag, the facet arrangement, and the facet orientation are important to capture where and how the gas is injected into the airbag. The inlet facets should be rigidly attached to the structure. This ensures that the inlet facets maintain their shape and size and undergo rigid body motion as the structure deforms and moves. You can use a BEAM MPC type to rigidly connect the nodes of the inlet facet to the structure.

For further details on defining inlet geometry and inlet blocking behavior of a particle generator, see *Particle Generator*.

Defining Inflator Mass Flow Rate and Temperature Data

The mass flow rate and temperature data are used to generate lumped molecules for a gas species. The mass flow rate and temperature of a gas species can be specified in tabular form. This form of specifying the mass flow rate and temperature for the inflator is identical to the uniform pressure method (see *Inflator Definition*).

The fluid inflator properties must be associated with a particle generator. The particle generator uses the mass flow rate to determine the incremental amount of gas that must be generated at any given time. The particle generator ensures that an equivalent number of particles are generated. The mass of a gas particle depends on the total amount of gas and the maximum number of particles requested. For further details on how the particle generator accounts for the incremental mass that must be generated, see *Particle Generator*.

Triggering the Inflator

There is usually a time lag between the start of the analysis and the deployment of the airbag in a crash simulation. In the LKM method, this time delay is introduced through an amplitude curve. The common form of such a curve is the step function. The area under the amplitude curve is the value of the time that the particle generator uses to look up the fluid inflator property data to compute the mass flow rate and the temperature of each gas species. The particle generator begins firing when the step-function curve becomes nonzero. You can use a constant unit amplitude curve for zero time delay. You can specify the inflation time amplitude for fluid inflator activation to trigger the inflator using this approach.

Another way to trigger the inflator is to use a generation time amplitude to define the particle generator flow. The inflator starts generating gas particles based on the specified amplitude definition. The inflation time amplitude

for fluid inflator activation replaces the generation time amplitude for particle generator flow if you specify both amplitudes in the model.

Input File Usage: Use the following options to activate a fluid inflator by specifying the inflation time amplitude:

**FLUID INFLATOR*, NAME=*inflator name*
**FLUID INFLATOR ACTIVATION*, INFLATION TIME AMPLITUDE=*inflator amplitude*
inflator name

Use the following options to activate a fluid inflator by specifying the generation time amplitude:

**FLUID INFLATOR*, NAME=*inflator name*
**PARTICLE GENERATOR FLOW*, GENERATION TIME AMPLITUDE=*inflator amplitude*
inflator name

Elements

The LKM method uses PD3D elements to model the lumped gas molecules. The actual elements are generated automatically during the analysis and appended to the element set associated with the discrete section. The properties of the specific gas species are associated with the elements by referring to the fluid behavior from the discrete section.

Abaqus/Explicit computes the size and mass of lumped molecules automatically. The density specified on the discrete section definition is ignored in the LKM method. The mass and rotary inertia proportional damping value on the discrete section definition are also ignored.

Usually, gas particles are contained, but sometimes a few particles can escape and cause numerical problems. To deactivate contact between particles that leak out of the airbag with the surrounding structures, you can specify the coordinates of the lower left corner and upper right corner of a control box. You should ensure that the control box is large enough to accommodate the fully deployed airbag including any motion of the airbag. All collisions of particles outside the control box are ignored. You can specify the name of the section control on the discrete section to associate the control box with the airbag.

Input File Usage: Use the following options to define lumped molecules:

**DISCRETE SECTION*, ELSET=*element set name*, FLUID BEHAVIOR=*fluid behavior name*,
CONTROLS=*section control name*
blank data line
**FLUID BEHAVIOR*, NAME=*fluid behavior name*
**SECTION CONTROLS*, NAME=*section control name*
blank data line
blank data line
cxl, cyl, czl, cxu, cyu, czu, 1

Switching from Lumped Kinetic Molecular Method to Uniform Pressure Method

During the early phase of deployment, the pressure inside the airbag is nonuniform. The LKM method can capture such nonuniformity of pressure inside the airbag. During the early phases of deployment, it is recommended that you use the LKM method. As the bag approaches full deployment, the pressure inside the airbag tends toward a uniform value. The uniform pressure method (UPM) is computationally efficient for such situations.

Therefore, for computational efficiency, switching from LKM to UPM when the airbag has nearly deployed is desirable. You can specify the time when Abaqus/Explicit switches from LKM to UPM. At the point of switching, the particles are deactivated and their motion is frozen. You should exercise caution when specifying the switching time. Switching too early when the pressure in the airbag is nonuniform can result in an inaccurate solution, while switching too late sacrifices computational efficiency. Another important reason to switch from LKM to UPM is to account for leakage of gas via vents and fabric of the airbag. You can activate the exchange of fluid between the airbag and its environment at the same time as the switching occurs. For more information, see *Fluid Exchange Definition*.

Input File Usage: Use the following options to switch from the lumped kinetic molecular method to the uniform pressure method:

**FLUID CAVITY*, SURFACE=*bag surface*, SWITCHTIME=*time to switch from LKM to UPM*
**FLUID EXCHANGE*, NAME=*fluid exchange name*, SURFACE=*bag surface*, PROPERTY=*fluid exchange property name*
**FLUID EXCHANGE PROPERTY*, NAME=*fluid exchange property name*
...
**FLUID EXCHANGE ACTIVATION*, AMPLITUDE=*name of amplitude*
fluid exchange name

Interactions

In the LKM method particles collide elastically with each other, as well as with the surrounding structure. All collisions preserve the momentum, as well as the energy of the particles. Because contact interactions form the basis for the LKM method, the general contact definition for LKM is very similar to the general contact definition for a discrete element method model. Each LKM particle is typically involved in the following contact interactions:

- Contact with another particle from the same gas species.
- Contact with a structural facet.

Separate element-based surfaces spanning particles of each gas species must be used to list each of the above contact interactions. A surface interaction must be defined for each of the contact interactions involving the particles. A special pressure-overclosure type is required to ensure purely elastic collisions. Dissipative contact interactions such as friction and contact damping are ignored for the LKM method.

Abaqus/Explicit adjusts the time increment automatically to ensure that lumped molecules are tracked accurately.

Input File Usage: Use the following options to define lumped molecules:

**SURFACE*, NAME=gas_surf
**SURFACE*, NAME=wall_surf
...
**SURFACE INTERACTION*, NAME=lkm_inter
**SURFACE BEHAVIOR*, PRESSURE-OVERCLOSURE=LKM
,
...
**CONTACT*
**CONTACT INCLUSIONS*
gas_surf, gas_surf
gas_surf, wall_surf
**CONTACT PROPERTY ASSIGNMENT*
gas_surf, gas_surf, lkm_inter
gas_surf, wall_surf, lkm_inter

Output

The local pressure at any instant of time on a patch of elements is the sum of the particle impact forces divided by the area of the patch. Due to the discrete nature of the impacts, the local pressure is noisy. Therefore, filtering is recommended for any local pressure computation. In the LKM method the mass, average temperature, and average pressure of a gas species are of interest. Three integrated output variables (PDMASS, PDTEMP, and PDPAVG) can be used to request history output for mass, average temperature, and average pressure for a specific gas species.

Input File Usage: Use the following options to request output for a gas species:

**DISCRETE SECTION*, ELSET=*gas1*, FLUID BEHAVIOR=*fluid behavior name*
blank data line
**OUTPUT*, HISTORY
**INTEGRATED OUTPUT*, ELSET=*gas1*
PDMASS, PDTEMP, PDPAVG

Limitations

The following limitations apply:

- Only the particle generator can inject gas in the fluid cavity in the LKM method. Therefore, the initial gas cannot be modeled in the LKM method. To approximate the initial gas, the particle generator can be fired for a short duration and then halted at the start of the analysis before the main analysis.
- The LKM method cannot be used in conjunction with other particle methods, such as SPH or DEM.
- An LKM analysis with more than one interacting gas species is not supported.

Input File Template

```
*PHYSICAL CONSTANTS, UNIVERSAL GAS CONSTANT=R, BOLTZMANN=K
...
*FLUID CAVITY, REF NODE=flucavrefnode, SURFACE=airbag_surface, ADIABATIC,
BEHAVIOR=air_gas, AMBIENT PRESSURE=value of ambient pressure
...
*DISCRETE SECTION, ELSET=air_particles, FLUID BEHAVIOR=air_gas, CONTROLS=lkm ccontrol
blank data line
*SECTION CONTROLS, NAME=lkm control
blank data line
blank data line
cxl, cyl, czl, cux, cuy, cuz, 1
*FLUID BEHAVIOR, NAME=air_gas
*MOLECULAR WEIGHT
value of molecular weight
*CAPACITY, TYPE=polynomial
value of heat capacity coefficients separated by commas
...
*PARTICLE GENERATOR, NAME=inflator, MAXIMUM NUMBER OF PARTICLES=number of particles,
FLUID CAVITY REFNODE=flucavrefnode, CAVITY VOLUME=final volume of airbag
*PARTICLE GENERATOR INLET, SURFACE=name of inlet surface
...
*PARTICLE GENERATOR MIXTURE
air_particles
...
*FLUID INFLATOR PROPERTY, NAME=inflator prop, TYPE=TEMPERATURE AND MASS
time0, temp0, M0
time1, temp1, M1
...
*FLUID INFLATOR MIXTURE, NUMBER SPECIES=1, TYPE=MASS FRACTION
air
time0, 1.0
time1, 1.0
...
*SURFACE, NAME=gas_surf
*SURFACE, NAME=wall_surf
*SURFACE INTERACTION, NAME=lkm_inter
*SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=LKM
...
*STEP
*DYNAMIC, EXPLICIT, ELEMENT BY ELEMENT
 , total time
*PARTICLE GENERATOR FLOW, FLUID INFLATOR PROPERTY=inflator prop,
GENERATION TIME AMPLITUDE=gen time amp, FLOW AMPLITUDE=flow amp, MASS FLOW RATE TYPE=TOTAL
...
*CONTACT
*CONTACT INCLUSIONS
gas_surf, gas_surf
gas_surf, wall_surf
*CONTACT PROPERTY ASSIGNMENT
gas_surf, gas_surf, lkm_inter
gas_surf, wall_surf, lkm_inter
```

Sequentially Coupled Multiphysics Analyses

You can use a sequentially coupled multiphysics analysis when the coupling between one or more of the physical fields in a model is important only in one direction.

A common case is a sequential thermal-stress analysis, where volumetric strain arises in a stress analysis if thermal expansion is included, but the deformations computed in the stress analysis do not significantly alter the thermal response. In such cases, the external fields that are imported typically vary with position and may be time dependent. The fields are predefined because they are not changed by the current analysis.

When the coupling is strong, the fields of both models influence each other, and you should consider fully coupled procedures in Abaqus or the co-simulation solution technique to accurately solve your multiphysics problem.

In this section:

- *General Capability for Importing External Fields*
- *Predefined Fields for Sequential Coupling*
- *Sequentially Coupled Thermal-Stress Analysis*
- *Sequentially Coupled Injection Molding to Stress Analysis*
- *Predefined Loads for Sequential Coupling*

General Capability for Importing External Fields

Products: Abaqus/Standard Abaqus/Explicit

References:

- *The Output Database*
- **EXTERNAL FIELD*
- **FIELD IMPORT*
- **DISTRIBUTION*
- **DISTRIBUTION TABLE*
- **INITIAL CONDITIONS*
- **FIELD MAPPER CONTROLS*
- **ADJUST*
- **UNIT SYSTEM*

Overview

A general capability is available for importing external fields in a sequential analysis.

Introduction to Importing External Fields

You can import external fields from an output database (`.sim`) file, also referred to as a SIM database (see *The Output Database*), or from another output database for which a Reader process is available. You mal also develop your own Reader to import your custom data. Once imported, the external field can be used to define distributions (SIM database only), initial conditions (SIM database only), and history-dependent fields, such as loads, boundary conditions, and predefined fields. External fields can also be used for node-based submodeling as discussed in *About Submodeling*.

The external field can be evaluated at nodes, integration points, or element centers. Conservative mapping is performed automatically if the fields are evaluated at different locations or if the meshes are different. Importing an external field is supported for most two-dimensional and three-dimensional continuum elements. When used to define distributions and initial conditions, external fields are also supported for three-dimensional conventional shell elements made of a single material. All field quantities are exchanged in the global Cartesian coordinate system.

Abaqus offers alternate interfaces for sequential coupling (see *Predefined Fields for Sequential Coupling*, *Sequentially Coupled Thermal-Stress Analysis*, and *Predefined Loads for Sequential Coupling*). These interfaces are limited to importing temperatures, normalized concentrations, and electromagnetic loads of a previous Abaqus analysis.

Specifying the Output Database File to Be Read

External fields can be read from various supported output databases. A Reader, an executable designed to read a particular database format, is started by the Abaqus driver to read and transfer the field data. *Table 1* lists the Readers availble for Dassault Systemes' solvers.

You can import external fields from a SIM database (`.sim`) file to import distributions, initial conditions and history-dependent fields. By deafault, if no Reader is specified the SIM Reader is used. You must specify the SIMfile name (including the file extension) from which to import external fields.

Input File Usage: Use the following option to specify a SIM database:

**EXTERNAL FIELD*, FILE=*file name*.sim

You can import history-dependent fields from a supported non-SIM database, or develop your Reader to process your data files. You must set the READER parameter to the Reader executable name, and provide the database file name. Consult the respective solver manual to install and determine the reader executable name.

Table 1: Readers distributed by Dassault Systemes

Reader Name	Description	Application
SMACseSimReader	Reads a SIM output database	• Abaqus/Standard • Abaqus/Explicit • SIMULIA Navier-Stokes solver (available in the Fluid Scenario Creation app on the **3D**EXPERIENCE platform) • SIMULIAplastic injection solver (available in the Plastic Injection Engineer app on the **3D**EXPERIENCE platform)
CSTStidioReader	Reads selective fields from CST Studio output database	CST Studio Suite

Input File Usage: Use the following option to specify a non-SIM database file name and reader executable name:

**EXTERNAL FIELD*, READER=*reader executable name*, FILE=*file name*

Importing an External Field to Define Distributions

You can import external fields to define distributions (see *Defining Distributions by Importing Field Data from an Output Database File*). A common case is an injection molding manufacturing process where a molten fibrous material is injected into a mold. You can use injection molding software to model the manufacturing process and then import the section and material properties (such as local element orientation and fiber dispersion) into a subsequent stress-displacement simulation.

Input File Usage: Use the following options to import an external field to define a distribution:

**DISTRIBUTION*, TABLE=*distribution table name*
, field data
**EXTERNAL FIELD*, FILE=*file name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Importing an External Field to Adjust Nodal Coordinates

You can import an external field to adjust nodal coordinates.

One common case is to introduce a geometric imperfection into a model for a postbuckling analysis. In this case, the previous analysis must contain the mode shapes from an eigenfrequency extraction analysis or nodal displacements from a static or dynamic analysis. Mapping is performed if the postbuckling analysis uses a dissimilar mesh. The imperfection is introduced by adjusting the nodal coordinates using the imported modal shapes or nodal displacements.

Another common case is to adjust nodal coordinates using the deformed configuration from a previous analysis. In this case, the previous analysis must write the nodal displacements to a SIM database.

Input File Usage: Use the following options to import mode shapes to introduce a geometric imperfection:

**ADJUST*, NODE SET=*nset*, DISTRIBUTION=*distribution name*
**DISTRIBUTION*, NAME=*distribution name*, TABLE=*distribution table name*
, *field data*
**EXTERNAL FIELD*, FILE=*file name*, MODE=*mode number*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to import nodal displacements to adjust nodal coordinates:

**ADJUST*, NODE SET=*nset*, DISTRIBUTION=*distribution name*
**DISTRIBUTION*, NAME=*distribution name*, TABLE=*distribution table name*
, *field data*
**EXTERNAL FIELD*, FILE=*file name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Importing an External Field to Define Initial Conditions

You can import an external field to define initial conditions (see *Initial Conditions*). A common case is a geomechanics analysis where you import an initial pore pressure field. You can compute a pore pressure field using a reservoir code and then import the field into a subsequent geostatic stress state procedure (see *Geostatic Stress State*). This procedure is used to compute the equilibrium state before performing a geotechnical analysis using the coupled pore fluid diffusion and stress analysis procedure in Abaqus (see *Coupled Pore Fluid Diffusion and Stress Analysis*). In this case, the reservoir code must write the pore pressure to a SIM database.

Input File Usage: Use the following options to import an external field to define initial conditions:

**INITIAL CONDITIONS*
**EXTERNAL FIELD*, FILE=*file name*

Importing an External Field to Define History-Dependent Fields

You can import external fields to define loads, boundary conditions, and predefined fields. A common case is a sequential thermal-stress analysis, where you first perform a heat transfer analysis (see *Uncoupled Heat Transfer Analysis*) to compute a temperature field and then import the temperatures into a subsequent stress-displacement analysis (see *Static Stress Analysis*).

You can import external fields for the following time-based analysis procedures:

- *Static Stress Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Explicit Dynamic Analysis*
- *Uncoupled Heat Transfer Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Piezoelectric Analysis*
- *Coupled Thermal-Electrical Analysis*
- *Eddy Current Analysis*

- *Coupled Pore Fluid Diffusion and Stress Analysis*
- *Mass Diffusion Analysis*
- *Linear Elastic Fatigue Crack Growth Analysis*

External fields can be imported only in one analysis step per analysis. For any subsequent analysis steps, the applied external fields remain constant based on the last imported values.

Input File Usage: Use the following options to import an external field to define a load, boundary condition, or predefined field:

**FIELD IMPORT*
**EXTERNAL FIELD*, FILE=*file name*

Saving Fields in a Previous Analysis

If the previous analysis is an Abaqus analysis, you request the source fields to write to the output database (`.sim`) file. When importing external fields over a time range, you must request output at a frequency to achieve sufficient solution accuracy (see *Controlling the Frequency of Output to the Output Database*).

Identifying the Target and Source Fields

When importing an external field, you define the target and source fields. The target field is the field defined in the current analysis. The source field is the field computed in a previous analysis. A field is a physical quantity that spans a region. The field is a node-based field if evaluated at nodes or an element-based field if evaluated at element centroids, integration points, or surface facets. You define a region by specifying a node set, an element set, or a surface regardless of where the field is evaluated.

When a node-based field is imported and an element set from the previous model is specified, the field is extracted from nodes of the elements in the specified element set. Similarly, when a node-based field is imported and a surface from the previous model is specified, the field is extracted from the nodes belonging to the surface facets of the specified surface. When an element-based field is imported and a node set from the previous model is specified, the element field is extracted from elements containing the nodes in the node set. Elements connected to nodes that are not in the node set are not included.

When importing an external field, you specify the region name space, the region name, and the physics quantity for the target field and the source field. The region name space specifies whether the region is a node set, an element set, or a surface. The region name is the name of a node set, element set, or surface.

When specifying the target field and source field parameters, you should consider the following conditions:

- The source field region name is optional. If it is not specified, the entire model is used. However, you are encouraged to provide the source field region name to reduce the computational cost associated with mapping and data transfer. Abaqus has complex naming conventions, especially for assembly definitions. The source file name must correspond to the name written to the SIM results file when reading from a previous Abaqus analysis.
- When defining initial conditions, the target field region name space and physics quantities are defined by the initial condition definition and, consequently, are not used.
- When defining distributions, the target field region name space, region name, and physics quantities are defined by the distribution definition and, consequently, are not used.
- When used for node-based submodeling, the target region is a node set.

The physics quantities for the target field and the source field are defined using identifiers. The physics quantity identifiers for the target field depend on the analysis procedure and are the same identifiers used for co-simulation for imported fields. *Table 2* lists the supported identifiers, and detailed information is provided in *Identifying*

the Fields Exchanged across a Co-Simulation Interface). Most field-type quantities can be extracted from the output database file. If the source analysis is an Abaqus analysis, the identifiers are defined in *Abaqus/Standard output variable identifiers* and *Abaqus/Explicit Output Variable Identifiers*. Certain output variables are inconsistent between Abaqus solvers. For example, Abaqus/Standard uses the variable NT and Abaqus/Explicit uses the variable NT11 to output nodal temperatures. You should visualize the results database to obtain the appropriate field name.

Table 2: Supported target field identifiers.

Analysis	Physics Quantity ID	Description
Analysis procedures supporting mechanical degrees of freedom	U	Displacement
	V	Velocity
	CF	Concentrated force at a node
	CM	Concentrated moment at a node
	P[1, 2]	Surface pressure
	TRVEC[1, 2]	Surface traction vector
Analysis procedures supporting thermal degrees of freedom	NT	Nodal temperature
	CFL	Concentrated heat flux at a node
	HFL[1, 2]	Surface heat flux
Analysis procedures supporting pore fluid pressure degree of freedom	POR[1]	Pore pressure
	CFF	Concentrated fluid flow at a node
Analysis procedures supporting mass concentration	CFL	Concentrated mass flux at a node
	SINV	Equivalent pressure stress
	NNC	Normalized concentration
Analysis procedures supporting electric potential	EPOT	Electric potential
	ECD	Electric current density
All procedures	FV*n*	Predefined field
	ESDV*n*[1]	Element solution-dependent variable
[1]Procedures excluding explicit dynamics.		
[2]When importing surface fields, the fields must be written at surfaces in the `.sim` file. Abaqus does not support a capability to write fields at surfaces. Surface output to the `.sim` file is supported by the SIMULIA Navier-Stokes solver (**3D**EXPERIENCE platform Fluid Scenario Creation app).		

Fields are mapped automatically when the source and target meshes are topologically different. However, Abaqus performs a mesh check internally before importing distributions and initial conditions, and, if the following conditions are satisfied, no mapping is performed:

- All of the nodes and elements in the target region are found in the source output database (`.sim`) file with matching labels.
- Matching elements in the target region and the source `.sim` file have the same connectivities and element types.
- Source data are requested at the same location (nodes or element integration points) where distributions are specified.

Input File Usage: Use the following options to import external fields to define history-dependent fields:

**FIELD IMPORT*
**EXTERNAL FIELD*, FILE=*file name*

NODES or ELEMENTS or SURFACE, *target region*, *target quantity ID*, NODES or ELEMENTS or SURFACE, *source region*, *source quantity ID*

For example,

**FIELD IMPORT*
**EXTERNAL FIELD*, FILE=`thermalAnalysis.sim`, START TIME=0, END TIME=3600
ELEMENTS, `ESTRESS`, NT, ELEMENTS, `ETHERMAL`, NT

imports temperatures defined as nodal temperatures (NT) from output database `thermalAnalysis.sim` and applies them as a nodal temperature field in the current stress analysis. In this case the current analysis does not have temperature degrees of freedom; therefore, temperatures are imported as a predefined temperature field at the nodes. The temperatures are extracted from the body defined by element set `ETHERMAL` in the `thermalAnalysis` job and are mapped to the body defined by element set `ESTRESS` in the current analysis.

Importing Field Variables and Temperature as a Field Variable

When importing field variables and temperature as a field variable, you might notice differences when you compare the source and target fields. There are two possible causes of these differences:

- Field variables and temperature as a field variable are mapped accurately and imported at nodes. However, when saved to the output database, these quantities are interpolated to element integration points. When contouring, some averaging can occur depending on the contouring parameters selected.
- For fully integrated first-order elements, a selective reduced-integration technique is employed (see *Solid isoparametric quadrilaterals and hexahedra*). In this case field variables and temperature as a field variable are averaged at the element center and saved to the output database.

The values correctly reflect the internal magnitudes used by Abaqus.

Importing Fields at a Specified Step and Increment

If you are reading from a SIM database to define distributions or initial conditions, you can specify the step and the increment from which to extract the field values. You must ensure that the field is saved to the SIM database at the specified step and increment. If the increment is omitted, the values at the end of the step are imported.

Input File Usage: Use the following option to import field values at a specified step and increment of a previous analysis to define distribution or initial conditions:

**EXTERNAL FIELD*, STEP=*analysis step*, INC=*increment*

Use the following option to import field values at the end of a specified step to define distribution or initial conditions:

**EXTERNAL FIELD*, STEP=*analysis step*

Importing Fields at a Snap-Shot Point

You can import fields at a specified snap-shot point. For a time-domain analysis, a snap-shot point is a specified time. For a frequency-domain analysis, a snap-shot point is a specified frequency or mode number. Frequency-domain results can be imported only from a non-SIM database. If the snap-shot point does not coincide

with when the fields were saved, Abaqus linearly interpolates the field values between the snap-shot points available to obtain the field values at the request time or frequency.

You can specify the analysis step. In this case the time refers to the step time to extract field values in the output database, Otherwise, time refers to the total time.

To import the last saved values of an analysis step or of the analysis, do not specify any of the time or frequency parameters. This might be useful when reading from a linear perturbation step or from a steady-state analysis where time has no physical meaning.

Input File Usage: Use the following option to import fields at a specified time (specified as total time):

**EXTERNAL FIELD*, TYPE=*SNAP SHOT*, FILE=*file name*, TIME=*total time*

Use the following option to import fields at a specified analysis step and time (specified as step time):

**EXTERNAL FIELD*, TYPE=*SNAP SHOT*, STEP=*analysis step*, TIME=*step time*

Use the following option to import fields at the end of an analysis step:

**EXTERNAL FIELD*, TYPE=*SNAP SHOT*, STEP=*analysis step*

Use the following option to import fields at the end of an analysis:

**EXTERNAL FIELD*, TYPE=*SNAP SHOT*

Use the following option to import fields at a specified frequency:

**EXTERNAL FIELD*, TYPE=*SNAP SHOT*, FREQ=*frequency*

Use the following option to import fields from a specified mode:

**EXTERNAL FIELD*TYPE=*SNAP SHOT*, MODE=*mode number*

Importing Fields over a Specified Time Range

You can import fields over a specified time range when importing fields to define history-dependent fields. You specify the start time and end time to define the time range in the upstream analysis to import fields.

Typically, data corresponding to the time points (increments of the downstream analysis) are not present in the output database. In this case field values are linearly interpolated between the time points stored in the output database to obtain values at the required time. Because the interpolation is linear, you must ensure that sufficient data is available in the output database to achieve sufficient accuracy.

If you are reading from a SIM database, you can specify the analysis step. In this case the time range refers to the step time to extract field values in the output database. Otherwise, time refers to the total time.

To import results over an analysis step or the entire analysis, do not specify a start and end time. If you omit the end time, results are imported as follows:

- If you specified an analysis step, results are imported to the end of the analysis step.
- If you omitted the analysis step, results are imported to the end of the analysis.

Input File Usage: Use the following option to import fields over a specified time range (specified as total time):

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*, START TIME=*start time*, END TIME=*end time*

Use the following option to import fields over a specified time range (specified as step time) in an analysis step:

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*, STEP=*analysis step*, START TIME=*start time*, END TIME=*step end time*

Use the following option to import fields over the entire upstream analysis:

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*

Use the following option to import fields over a time range, starting at a specified time to the end of the analysis:

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*, START TIME=*start time*

Use the following option to import fields over a time range in a particular analysis step, starting at a specified time to the end of the analysis step:

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*, STEP=*analysis step*, START TIME=*start time*, END TIME=*step end time*

Resolving Different Physics Time Scales

When importing fields over a time range, Abaqus performs no time scaling by default. Specifically, if the upstream analysis time range is larger than the downstream analysis time range, Abaqus imports only the partial upstream fields. When the upstream analysis is shorter than the downstream analysis, Abaqus keeps the last imported field values constant for the remaining duration of the downstream analysis step.

For certain multiphysics simulations, the physics time scales might be different when performing a sequential analysis. For example, you can perform an electromagnetic analysis followed by a thermal-stress analysis importing heat losses. The time scale for the electromagnetic analysis might be significantly shorter than the time scale for conduction in the thermal-stress analysis. When the requested time range of the upstream analysis is different from the analysis step time of the downstream analysis, you can scale time such that the requested upstream analysis range is mapped over the entire downstream analysis step. In this case you must provide an end time value explicitly.

Input File Usage: Use the following option to map the upstream analysis results over the downstream analysis step:

**EXTERNAL FIELD*, FILE=*file name*, TYPE=*TIME RANGE*, TIME SCALING=*ON*, START TIME=*start time*, END TIME=*end time*

Translation Between Different Units Systems

By default, Abaqus assumes consistent units between the Abaqus model and the previous analysis model; therefore, no units translation is necessary. If the Reader process supports units system and you have declared the units system in the Abaqus model (see *Units*), the appropriate translation for mesh coordinates and field data is performed when field data are imported. This translation can be performed only for field types that have a predefined dimensionality (for example, the velocity field has a predefined dimensionality of $LT^{\{-1\}}$). Field

variables in Abaqus do not have a predefined dimensionality; therefore, unit translation for field variables is not performed. In this case you can scale the field variable by applying a field operation.

Applying Field Mapper Controls

Typically, the source and target regions have different meshes. Fields are mapped conservatively between the source and target meshes, and Abaqus selects the most appropriate mapping algorithm based on the field properties. For most cases, the default settings for the mapping algorithm are adequate, but you can modify the mapper settings by specifying field mapper controls when importing distributions and initial conditions.

Missing members might be reported during the mapping operation. Missing members are nodes or elements for which the mapper cannot find a correspondence between source and target meshes. Consequently, the mapper does not compute field values at these locations. In this case, you can modify the default search tolerances or specify how to treat missing members. A zero value is applied automatically for missing members when importing history-dependent flux quantities (that is, forces, pressure, tractions, and heat fluxes) and no treatment is required.

Input File Usage: Use the following options to specify field mapper controls:

**FIELD MAPPER CONTROLS*, NAME=*field mapper controls name*
**EXTERNAL FIELD*, FILE=*file name*
, , , , , , *field mapper controls name*

Applying Field Operations

You can apply one or more field operations when importing fields.

You can scale a field by a constant value by providing a scale factor. The scaling factor is intended to scale a field by a physical constant. It is not intended for unit conversion; for example, where you might need additional parameters to convert a temperature field.

You can extract a component of a vector or tensor field by providing a component index, i. The i^{th} component of the source vector or tensor field is extracted and then mapped and imported as a scalar field. The components are extracted in the local coordinate system of the source field. You can apply component extraction when importing distributions, initial conditions, and history-dependent fields; it cannot be used for a submodel analysis.

If a region in the previous analysis is not colocated with the region in the current analysis, you must specify the translation and rotation to reposition the source region before field values are imported. You specify a rotation by giving two points, a and b, to define a rotation axis and a right-handed angular rotation about the axis. Local coordinate systems defined within the source region are translated and rotated according to the specified positioning data.

Input File Usage: Use the following options to scale a field by a constant value:

**FIELD OPERATIONS*, NAME=*name*, SCALE FACTOR=*scale factor*
**EXTERNAL FIELD*, FILE=*file name*
, , , , , , , *field operation name*

Use the following options to extract the i^{th} component of the source field:

**FIELD OPERATIONS*, NAME=*name*, COMPONENT=*index*
**EXTERNAL FIELD*, FILE=*file name*
, , , , , , , *field operation name*

Use the following options to reposition the source region with the target region before importing data:

**FIELD OPERATIONS*, NAME=*field operation name*
x, *y*, *z*
x1, *y1*, *z1*, *x2*, *y2*, *z2*, *rotation*
**EXTERNAL FIELD*, FILE=*file name*
, , , , , , , *field operation name*

Limitations

The following limitations apply:

- Volumes or surfaces defined over beam, pipe, and truss elements or edges of shell or membrane elements cannot be used to specify the target region. When appropriate, you should import values at nodes.
- Source or target regions with the following underlying element types are not supported: modified triangular elements, modified tetrahedral elements, second-order temperature-displacement elements, generalized plane strain elements, and axisymmetric shells.
- Tensor fields cannot be imported from a source region with two-dimensional elements.
- The use of nonunique nodal values (for example, temperature and field variables) on shell elements from an Abaqus-generated SIM file using field import is not supported.
- Reading fields from non-SIM databases is supported only when importing history-dependent fields; it is not supported when importing distributions and initial conditions.
- When importing fields from the same results database using multiple **EXTERNAL FIELD* options, you must import all fields at a specific time or over a time range; in either case, the time or the time range must be the same.

Because importing external fields in a sequential analysis is a general capability, no error message is issued if Abaqus encounters the conditions listed above.

Predefined Fields for Sequential Coupling

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

Overview

The time history of the following nodal output quantities, generated in an Abaqus/Standard analysis, can be read into subsequent Abaqus/Standard analyses as predefined fields for sequentially coupled multiphysics workflows:

- Temperature
- Normalized concentration
- Electric potential

Alternatively, the time history of a known pore fluid pressure field can be read into a subsequent Abaqus/Standard static or an Abaqus/Explicit dynamic stress analysis as a predefined field to perform a sequentially coupled pore pressure–stress analysis.

A sequentially coupled multiphysics analysis can be used when the coupling between one or more of the physical fields in a model is only important in one direction—a special common case is a sequential thermal-stress analysis (*Sequentially Coupled Thermal-Stress Analysis*). While the uncoupled thermal-stress analysis is the most common sequential multiphysics workflow, the predefined field capability in Abaqus/Standard directly supports similar sequential workflows involving normalized concentrations (*Mass Diffusion Analysis*) and electric potentials (*Coupled Thermal-Electrical Analysis*). As with temperatures, normalized concentrations and electric potentials can be read from the output database (`.odb`) file into subsequent analyses as predefined fields.

When defined by results from a previous analysis, predefined fields typically vary with position and are time dependent—they are predefined because they are not changed by the current analysis. When predefined fields are read from a previous analysis, they are read in at the nodes. They are then interpolated within elements as needed (see *Interpolating Data between Meshes*). Any number of predefined fields can be read in, and material properties can be defined to depend on them. In addition, volumetric strain will arise in a stress analysis if thermal expansion (*Thermal Expansion*) or field expansion (*Field Expansion*) are included in the material property definition.

Predefined fields may affect the system response through:

- the constitutive behavior, such as the yield stress defined as a function of temperature or field variables; or
- volumetric strains when thermal or field expansion behaviors (*Thermal Expansion* and *Field Expansion*) are included in the material definition in a stress/displacement analysis; or
- a modification to the stress field when a pre-computed pore fluid pressure field is read into an Abaqus stress analysis as a predefined field variable.

References:

- *Analysis Procedures*
- *Defining an Analysis*
- *Sequentially Coupled Thermal-Stress Analysis*
- *Continuum Elements*
- *Predefined Fields*
- *Creating and modifying output requests*
- *Defining a temperature field*

Saving Temperatures, Normalized Concentrations, and Electric Potentials for Predefined Fields in Subsequent Analyses

Nodal temperatures, normalized concentrations, and electrical potentials can be stored as functions of time for use in subsequent analyses. Temperatures can be stored in either the results (`.fil`) file or the output database (`.odb`) file, but normalized concentrations and electrical potentials can be used only if they are stored in the output database file. Saved values must be read into the new analyses as predefined fields. See *Node Output* and *Writing Nodal Output to the Output Database*.

Saving Temperatures for Predefined Fields in Subsequent Analyses

To be read as a predefined field, nodal temperatures must be stored as functions of time in the results (`.fil`) file or output database (`.odb`) file. You can request nodal temperature output (NT) in an uncoupled heat transfer analysis or in a coupled thermal-electrical analysis.

Saving Normalized Concentrations for Predefined Fields in Subsequent Analyses

To be read as predefined fields, normalized concentrations must be stored as functions of time in the output database (`.odb`) file—unlike nodal temperatures they cannot be read directly from a results file. You can request nodal normalized concentrations output (NNC) in a mass diffusion analysis.

Saving Electric Potentials for Predefined Fields in Subsequent Analyses

To be read as predefined fields, electrical potentials must be stored as functions of time in the output database (`.odb`) file—unlike nodal temperatures they cannot be read directly from a results file. You can request nodal electric potential output (EPOT) in a coupled thermal-electrical analysis or a piezoelectric analysis.

Transferring Temperatures as Temperature Fields

To define the temperature field at different times in the current analysis, you read the nodal temperatures stored as a function of time in the heat transfer results or output database file. Nodes can be removed for the current problem; for example, in a sequential thermal-stress analysis elements that represent nonstructural parts of the heat transfer mesh (such as insulation or cooling fluid) can be omitted in the stress analysis. When the heat transfer results file or output database file is read, temperatures at nodes that are not present in the mesh for the current analysis are ignored.

You must specify the name of the thermal analysis results file or output database file that contains the required nodal temperatures. The file extension is optional. If the heat transfer model and the current analysis model share the same mesh, the default is the results file. If the heat transfer model and the current analysis model have dissimilar meshes, the output database file must be used. See *Reading the Values of a Field from a User-Specified File* for more information.

If both models contain part and assembly definitions, the part (`.prt`) files from both analyses are required to transfer temperatures from the thermal analysis to the current analysis. If the thermal model is defined in terms of an assembly of part instances, the current analysis must be as well. The part instance names and local node numbers must be the same in both analyses for the nodes at which temperatures are transferred.

Transferring Temperatures, Normalized Concentrations, and Electric Potentials from the Output Database to Predefined Fields

To define predefined fields at different times in the current analysis, you can read nodal temperatures, normalized concentrations, or electric potentials stored as a function of time in the output database file. Nodes can be removed for the current problem. When the nodal output variables on the output database file are on nodes that are not present in the mesh for the current analysis, they are ignored.

You must specify the name of the output database file that contains the required nodal output variables as well as the nodal output label (NT, NNC, or EPOT) to identify the field that is being read. See *Defining Fields Using Nodal Scalar Output Values from a User-Specified Output Database File*.

If both models contain part and assembly definitions, the part (`.prt`) files from both analyses are required to transfer nodal results from the original analysis to the current analysis. If the original model is defined in terms of an assembly of part instances, the current analysis must be as well. The part instance names and local node numbers must be the same in both analyses for the nodes at which nodal results are transferred.

Transferring Pore Fluid Pressure to a Predefined Field

In some geotechnical applications a fully coupled pore fluid diffusion and stress analysis can be approximated by a sequentially coupled approach that consists of two parts:

1. A purely pore fluid flow analysis that computes the pore fluid pressure distribution in the porous medium without accounting for the deformation in the solid skeleton.
2. A pure stress analysis that only computes the deformation of the solid skeleton, taking into account the effects of the pore fluid pressure field computed in the first part.

In such an approach, you must transfer a known pore fluid pressure field (computed in the first part of the sequential workflow) into a stress analysis (the current analysis, which is the second part of the sequential workflow).

To define the pore fluid pressure field at different times in the current analysis, you must first associate the pore fluid pressure field with a specific field variable (see *Pore Fluid Pressure*). Subsequently, you can use any of the available methods to read or define predefined fields to include the known pore pressure field in the stress analysis.

Initial Conditions

Appropriate initial conditions for Abaqus/Standard procedures are discussed in *Analysis Procedures*. You can read the nodal temperatures, normalized concentrations, or electric potentials from previous analyses to initialize predefined fields. You can also read the stress, plastic strain, and damage initiation criteria from previous analyses to initialize predefined fields. When data are read from the output database (`.sim`) file, mapping is performed automatically between dissimilar meshes. See *Initial Conditions* for details.

Boundary Conditions

Appropriate boundary conditions for Abaqus/Standard procedures are discussed in *Analysis Procedures*. See also *Boundary Conditions*.

Loads

Appropriate loadings for Abaqus/Standard procedures are discussed in *Analysis Procedures*. See also *About Loads*.

Predefined Fields

See *Predefined Fields* for additional details on predefined temperatures and fields.

Material Options

See *Abaqus Materials Guide* for details on the material models available in Abaqus/Standard.

Volumetric strain will arise in a stress analysis if thermal expansion (*Thermal Expansion*) or field expansion (*Field Expansion*) is included in the material property definition.

Elements

Continuum and structural elements available in Abaqus/Standard are discussed in *Continuum Elements*Continuum Elements and *Structural Elements*. Details on how results from a previous analysis can be transferred to a current analysis are discussed in *Predefined Fields*.

Output

Appropriate output variables for Abaqus/Standard are described in *Abaqus Materials Guide*. All of the output variables are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

Two examples of sequential workflows are shown.

The first example is a sequentially coupled moisture-stress analysis, which is an example of a sequentially coupled multiphysics analysis. A typical sequentially coupled moisture-stress analysis consists of two Abaqus/Standard runs: a mass diffusion analysis and a subsequent stress analysis. Normalized concentrations are stored in the output database file for the mass diffusion analysis and read into the subsequent stress analysis as a predefined field. An example problem is shown in *Reading scalar nodal output from the output database into field variables*.

The following template shows the input for the mass diffusion analysis `massdiffusion2d.inp`:

```
*HEADING
…
*ELEMENT, TYPE=DC2D4
(Choose the mass diffusion element type)
…
*STEP
*MASS DIFFUSION
…
Apply loads and boundary conditions
…
** Write all normalized concentrations to the output
** database file, massdiffusion2d.odb
*OUTPUT, FIELD
*NODE OUTPUT
 NNC
*END STEP
```

The following template shows the input for the subsequent static structural analysis:

```
*HEADING
…
*ELEMENT, TYPE=CPE4R
(Choose the continuum element type compatible with the mass diffusion element type used)
…
*MATERIAL
*EXPANSION, FIELD=1
(Define field expansion for field 1 so that the normalized concentration causes volumetric
strain in the stress analysis)
…
*STEP
*STATIC
```

```
…
Apply structural loads and boundary conditions
…
*FIELD, FILE=massdiffusion2d.odb, OUTPUT VARIABLE=NNC, VARIABLE=1
Read in all normalized concentrations from the output database file into field variable 1
…
*END STEP
```

The second example shows a template for including a known pore fluid pressure field as a predefined field variable in a static or in an explicit dynamic step.

```
*HEADING
…
*ELEMENT, TYPE=element_type
(Choose the appropriate type of continuum element to carry out the stress analysis)
…
*MATERIAL
*PORE FLUID PRESSURE, FIELD=n
Associate pore fluid pressure field with the predefined field variable n
…
*STEP
*STATIC (or *DYNAMIC, EXPLICIT)
…
Apply structural loads and boundary conditions
…
*FIELD, VARIABLE=n
Read in known pore fluid pressure field into predefined field variable n
…
*END STEP
```

Sequentially Coupled Thermal-Stress Analysis

Products: Abaqus/Standard Abaqus/CAE

Overview

A sequentially coupled heat transfer analysis:

- is used when the stress/deformation field in a structure depends on the temperature field in that structure, but the temperature field can be found without knowledge of the stress/deformation response; and
- is usually performed by first conducting an uncoupled heat transfer analysis and then a stress/deformation analysis.

A thermal-stress analysis in which the temperature field does not depend on the stress field is a common example of a sequential multiphysics workflow and is one case of the more general workflow described in *Predefined Fields for Sequential Coupling*. In such thermal-stress analyses, temperature is calculated in an uncoupled heat transfer analysis (*Uncoupled Heat Transfer Analysis*) or in a coupled thermal-electrical analysis (*Coupled Thermal-Electrical Analysis*).

References:

- *Defining an Analysis*
- *About Heat Transfer Analysis Procedures*
- *Predefined Fields for Sequential Coupling*
- *Creating and modifying output requests*
- *Defining a temperature field*

Saving the Nodal Temperatures

Nodal temperatures are stored as a function of time in the heat transfer results (`.fil`) file or output database (`.odb`) file by requesting output variable NT as nodal output to the results or output database file. See *Node Output* and *Writing Nodal Output to the Output Database*.

Transferring the Heat Transfer Results to the Stress Analysis

The temperatures are read into the stress analysis as a predefined field; the temperature varies with position and is usually time dependent. It is predefined because it is not changed by the stress analysis solution. Such predefined fields are always read into Abaqus/Standard at the nodes. They are then interpolated to the calculation points within elements as needed (see *Interpolating Data between Meshes*). The temperature interpolation in the stress elements is usually approximate and one order lower than the displacement interpolation to obtain a compatible variation of thermal and mechanical strain. Any number of predefined fields can be read in, and material properties can be defined to depend on them.

For more information, see *Transferring Temperatures as Temperature Fields*.

Initial Conditions

Appropriate initial conditions for the thermal and stress analysis problems are described in the heat transfer and stress analysis sections—for example, see *About Heat Transfer Analysis Procedures*, *Coupled Thermal-Electrical Analysis*, *About Static Stress Analysis Procedures*, and *About Dynamic Analysis Procedures*. See also *Initial Conditions*.

Boundary Conditions

Appropriate boundary conditions for the thermal and stress analysis problems are described in the heat transfer and stress analysis sections—for example, see *About Heat Transfer Analysis Procedures*, *Coupled Thermal-Electrical Analysis*, *About Static Stress Analysis Procedures*, and *About Dynamic Analysis Procedures*. See also *Boundary Conditions*.

Loads

Appropriate loading for the thermal and stress analysis problems is described in the heat transfer and stress analysis sections—for example, see *About Heat Transfer Analysis Procedures*, *Coupled Thermal-Electrical Analysis*, *About Static Stress Analysis Procedures*, and *About Dynamic Analysis Procedures*. See also *About Loads*.

Predefined Fields

In addition to the temperatures read in from the heat transfer analysis, user-defined field variables can be specified; these values only affect field-variable-dependent material properties, if any. See *Predefined Fields*.

Material Options

The materials in the thermal analysis must have thermal properties such as conductivity defined (see *About Thermal Properties*). Any mechanical properties such as elasticity will be ignored in the thermal analysis, but they must be defined for the stress analysis procedure. See *Abaqus Materials Guide* for details on the material models available in Abaqus/Standard.

Thermal strain will arise in the stress analysis if thermal expansion (*Thermal Expansion*) is included in the material property definition.

Elements

You can use any of the heat transfer elements in Abaqus/Standard in the thermal analysis. In the stress analysis, you must choose the corresponding continuum or structural elements. For example, if heat transfer shell element type DS4 is defined by nodes 100, 101, 102, and 103 in the heat transfer analysis, three-dimensional shell element type S4R or S4R5 must be defined by these nodes in the stress analysis procedure.

For continuum elements you can transfer heat transfer results from a mesh using first-order elements to a stress analysis with a mesh using second-order elements. For more information, see *Using Second-Order Stress Elements with First-Order Heat Transfer Elements (the Midside Node Capability)*. You can activate the interpolation capability to map nodal temperatures between dissimilar meshes of matching element dimensionality (solid elements to solid elements or shell elements to shell elements). You cannot map nodal temperatures between heat transfer shell elements and structural membrane elements using the interpolation capability. For more information, see *Interpolating Data between Meshes*.

Output

The nodal temperatures must be written to the heat transfer analysis results or output database file by requesting the output variable NT (see *Output to the Data and Results Files*). These temperatures will be read into the stress analysis procedure.

Appropriate output variables are described in the heat transfer and stress analysis sections. All of the output variables are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

A typical sequentially coupled thermal-stress analysis consists of two Abaqus/Standard runs: a heat transfer analysis and a subsequent stress analysis.

The following template shows the input for the heat transfer analysis `heat.inp`:

```
*HEADING
…
*ELEMENT, TYPE=DC2D4
(Choose the heat transfer element type)
…
*STEP
*HEAT TRANSFER
…
Apply thermal loads and boundary conditions
…
** Write all nodal temperatures to the results or
** output database file, heat.fil/heat.odb
*NODE FILE, NSET=NALL
 NT
*OUTPUT, FIELD
*NODE OUTPUT, NSET=NALL
 NT
*END STEP
```

The following template shows the input for the subsequent static structural analysis:

```
*HEADING
…
*ELEMENT, TYPE=CPE4R
(Choose the continuum element type compatible with the heat transfer element type used)
…
*STEP
*STATIC
…
Apply structural loads and boundary conditions
…
*TEMPERATURE, FILE=heat
Read in all nodal temperatures from the results or output database file, heat.fil/heat.odb
…
*END STEP
```

Sequentially Coupled Injection Molding to Stress Analysis

Products: Abaqus/Standard

Overview

A sequentially coupled injection molding to stress analysis:

- is used when the composite material is enforced with chopped fiber, and the fiber orientation contributes to the material response in the subsequent stress analysis; and
- is used when the residual temperature and stress in the plastic part that occurred during the injection molding process contribute to the mechanical response of the part subject to service load.

A sequentially coupled injection molding to stress analysis typically transfers the following results from injection molding analysis to a stress analysis:

- Fiber dispersion characterized as a second-order orientation tensor.
- Residual temperature.
- Residual stress.

You can also import local coordinate systems with X-, Y-, and Z-axes representing the principal directions of the second-order orientation tensor. When you import a local system, tensor (such as stress) components are output in the local system.

Results files from third-party injection molding software must be converted to an output database (`.sim`) file before the results can be read by Abaqus/Standard as predefined fields.

References:

- *Analysis Procedures*
- *Continuum Elements*
- *Distribution Definition*
- *Initial Conditions*

Introduction to Injection Molding

Injection molding is a manufacturing process for producing parts by injecting molten material into a mold. Injection molding can be performed with many types of materials, most commonly thermoplastic polymers. The material, sometimes mixed with chopped fiber, is injected into a mold cavity at high pressure, where it cools and hardens to a desired shape.

There are many design considerations and possibilities for the injection molding process. For fiber-reinforced plastics (FRPs), the gate location where the material is injected into the mold determines how fibers are aligned in the molded part. In a subsequent stress analysis with appropriate load and boundary conditions, you can study how the fiber orientation affects the strength of the material. The design of the molding machine can also result in different residual temperature and stress in the molded part when taken out of the mold cavity. Special considerations are required to minimize the warpage of the part due to the residual stress from the manufacturing process. The warpage analysis can also be performed in a subsequent stress analysis.

Saving Fiber Orientation, Temperature, and Stress for Predefined Fields in Subsequent Stress Analyses

Injection molding results saved in the output database (`.sim`) file can be read into the subsequent stress analyses as predefined fields. Fiber dispersion results are stored as a second-order orientation tensor. If homogenized composite material properties are used in the subsequent analysis and if the properties are not isotropic, material orientations (usually principal directions of the orientation tensor) must be stored and read into the second analysis as distributions.

Stress and temperature also need to be stored in the output database (`.sim`) if they are to be read as predefined conditions.

Fiber orientation, temperature, and stress can be stored either at nodes, material points, or element centroids. When read as predefined fields, mapping is performed automatically if they are read at different locations or from a dissimilar mesh.

Transferring Fiber Orientations as Distributions

The orientation of the fiber mixed into the molded parts can significantly affect the mechanical response of the parts. If the composite material is modeled at the macrolevel with homogenized properties (see *Defining the Elasticity of a Short-Fiber Reinforced Composite*), the fiber orientations are read into the subsequent stress analysis as material orientations. If the composite material is modeled with multiscale material (see *Multiscale Material Modeling*), the fiber orientation tensor results are read into the stress analysis as part of the material definition. Homogenization is then performed during the stress analysis using the orientation tensor results.

Transferring Temperature Results

The temperatures are read into the stress analysis as initial conditions (see *Importing Data from an Output Database File*). Such fields are always read into Abaqus/Standard at the nodes. If temperatures are stored on a dissimilar mesh or at different locations such as material points or element centroids, mapping is performed automatically. You can define material properties such as elastic moduli and thermal expansion coefficients to depend on the temperature.

Transferring Stress Results

The stresses are read into the stress analysis as initial conditions (see *Importing Data from an Output Database File*). Such fields are always read into Abaqus/Standard at the material points. If stresses are stored on a dissimilar mesh or at different locations such as nodes or element centroids, mapping is performed automatically. Residual stresses can result in different responses of the molded part under service load compared to the case when residual stresses are ignored.

Initial Conditions

Appropriate initial conditions for Abaqus/Standard procedures are discussed in *Analysis Procedures*. You can also read the stress and temperature from previous injection molding analyses to initialize these fields. Mapping is performed automatically when data are read at different locations or from a dissimilar mesh. For more information, see *Initial Conditions*.

Boundary Conditions

Appropriate boundary conditions for Abaqus/Standard procedures are discussed in *Analysis Procedures*. For more information, see *Boundary Conditions*.

Loads

Appropriate loads for Abaqus/Standard procedures are discussed in *Analysis Procedures*. For more information, see *About Loads*.

Predefined Fields

For additional details on predefined temperatures and fields, see *Predefined Fields*.

Material Options

For details on the material models available in Abaqus/Standard, see the *Abaqus Materials Guide*.

Elements

Only results stored in three-dimensional continuum elements can be read as predefined fields. Injection molding results stored in three-dimensional continuum elements can be read only into three-dimensional continuum elements. Continuum elements available in Abaqus/Standard are discussed in *Continuum Elements*.

Output

Appropriate output variables for Abaqus/Standard are described in the *Abaqus Materials Guide*. All of the output variables are outlined in *Abaqus/Standard Output Variable Identifiers*.

Input File Template

The following template shows the input for the workflow when injection molding results are imported to a local system (the composite in this example is linear elastic and macro-level elastic moduli and fiber dispersion are specified):

```
*HEADING
...
*ELEMENT, TYPE=C3D8, ELSET=EALL
Choose the three-dimensional continuum element type supported for a multiscale material
...
*DISTRIBUTION TABLE, NAME=orient_table
COORD3D, COORD3D
*DISTRIBUTION, LOCATION=ELEMENT, TABLE=orient_table, NAME=distori
*EXTERNAL FIELD, FILE=plastic.sim
Read in material orientation results from the output database file plastic.sim
*ORIENTATION, NAME=ORI, TYPE=RECTANGULAR, DISPERSION
distori,
3, 0
distoritens
Define a local system by specifying both the axes and the fiber dispersion
*SOLID SECTION, ELSET=EALL, ORIENTATION=ORI, MATERIAL=COMPOSITE
Define a solid section using the orientation distribution and the fiber dispersion distribution from the previous
 analysis
The axes of the local system are the principal directions of the orientation tensor
computed by the injection molding simulation
...
*DISTRIBUTION TABLE, NAME=oritens_table
ORITENS
*DISTRIBUTION, LOCATION=ELEMENT, TABLE=oritens_table, NAME=distoritens
```

```
*EXTERNAL FIELD, FILE=plastic.sim
Read in fiber orientation tensor results from the output database file plastic.sim
…
*MATERIAL, NAME=COMPOSITE
Define a multiscale composite material
*ELASTIC,TYPE=TRANSVERSELY ISOTROPIC
Specify the homogenized elastic moduli of composite with aligned fiber.
These values are computed from the first-step homogenization as shown in
Figure 2
*FIBER DISPERSION
No data are specified in the data line, as the fiber dispersion specified in the section definition will be used in
the analysis
…
*INITIAL CONDITIONS, TYPE=TEMPERATURE
*EXTERNAL FIELD, FILE=plastic.sim
Read in temperature results from the output database file plastic.sim
*INITIAL CONDITIONS, TYPE=STRESS
*EXTERNAL FIELD, FILE=plastic.sim
Read in stress results from the output database file plastic.sim
…
*STEP
*STATIC
…
Apply structural loads and boundary conditions
…
*END STEP
```

The following template shows the input for the workflow when injection molding results are imported to a global system (composite is nonlinear):

```
*HEADING
…
*ELEMENT, TYPE=C3D8, ELSET=EALL
Choose the three-dimensional continuum element type supported for a multiscale material
…
*ORIENTATION, NAME=ORI, TYPE=RECTANGULAR
1, 0, 0, 0, 1, 0
3, 0
*SOLID SECTION, ELSET=EALL, ORIENTATION=ORI, MATERIAL=COMPOSITE
Define a solid section using the composite material and a global coordinate system
…
*DISTRIBUTION TABLE, NAME=oritens_table
ORITENS
*DISTRIBUTION, LOCATION=ELEMENT, TABLE=oritens_table, NAME=distoritens
*EXTERNAL FIELD, FILE=plastic.sim
Read in fiber orientation tensor results from the output database file plastic.sim
…
*MATERIAL, NAME=COMPOSITE
*MEAN FIELD HOMOGENIZATION
*CONSTITUENT, TYPE=MATRIX
…
*CONSTITUENT, TYPE=INCLUSION
0.3, 20.0, distoritens
Define geometric attributes of the fiber including volume fraction, aspect ratio,
and fiber orientation tensor distribution
…
*INITIAL CONDITIONS, TYPE=TEMPERATURE
*EXTERNAL FIELD, FILE=plastic.sim
Read in temperature results from the output database file plastic.sim
```

```
*INITIAL CONDITIONS, TYPE=STRESS
*EXTERNAL FIELD, FILE=plastic.sim
Read in stress results from the output database file plastic.sim
…
*STEP
*STATIC
…
Apply structural loads and boundary conditions
…
*END STEP
```

Predefined Loads for Sequential Coupling

Products: Abaqus/Standard

References:

- *Mapping Thermal and Magnetic Loads*
- *Defining an Analysis*
- *Eddy Current Analysis*
- *Concentrated Loads*

Overview

The values of the following whole element output quantities, generated in an Abaqus/Standard time-harmonic eddy current analysis, can be read into subsequent Abaqus/Standard analyses as point loads for sequentially coupled multiphysics workflows:

- Rate of Joule heat dissipation
- Magnetic body force intensity

A sequentially coupled multiphysics analysis can be used to apply electromagnetically generated loads (from a time-harmonic eddy current analysis) in a heat transfer, coupled temperature-displacement, or stress/displacement analysis. In many cases coupling is important only from the time-harmonic eddy current analysis; the impact of loading on the structure's mechanical or thermal response is not great enough to affect the validity of the original time-harmonic eddy current analysis.

Saving Joule Heat Dissipation or Magnetic Body Force Intensity for Use in Subsequent Analyses

You can request Joule heat dissipation output (EMJH) or magnetic body force intensity output (EMBF) in a time-harmonic eddy current analysis. Only values stored in the output database (`.odb`) file are available for use with sequential coupling.

Converting Results for Subsequent Use

The whole element quantities are converted to nodal load quantities using the **abaqus emloads** utility. The utility converts Joule heat dissipation output to concentrated heat flux and magnetic body force intensity output to point loads. This utility also enables conversion of results between dissimilar meshes. For more information, see *Mapping Thermal and Magnetic Loads*.

Conversion Limitations

When converting results values between dissimilar meshes, global conservation of the net flux is ensured provided that the model domain in the heat transfer, coupled temperature-displacement, or stress/displacement analysis matches the model domain in the time-harmonic eddy current analysis. The conservative mapping algorithm used in the **abaqus emloads** utility also provides a locally smooth distribution of point flux values (either body force or concentrated heat flux) in cases where the mesh in the time-harmonic eddy current analysis is finer than the “target” representative mesh. In situations where this is not the case and the “target” representative mesh is finer or of similar size to the mesh in the time-harmonic eddy current analysis, you may observe nodal locations with zero converted flux values. In these cases you will still observe global conservation of the flux, but your

solution may be adversely affected locally. You can correct for these situations by always performing the time-harmonic eddy current analysis with a finer mesh.

Transferring Nodal Loads from the Output Database to Concentrated Loads

To define loads in a heat transfer, coupled temperature-displacement, or stress/displacement analysis, you can read nodal concentrated heat fluxes and point loads from the output database (`.odb`) file created by the **abaqus emloads** utility.

Input File Template

In this example heat flux values are stored in the output database from a time-harmonic eddy current analysis. These values, after conversion to point heat fluxes, are read into a subsequent analysis as a concentrated flux.

The following template shows the input for the time-harmonic eddy current analysis `electromagnetic.inp`:

```
*HEADING
…
*ELEMENT, TYPE=EMC3D8
(Choose the electromagnetic element type)
…
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
…
Apply loads and boundary conditions
…
** Write element Joule heat dissipation results to the output
** database file, electromagnetic.odb
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=CONDUCTOR
 EMJH
*END STEP
```

The following template shows the input for the heat transfer analysis, `heattransfer.inp`, which refers to an output database, `pointflux.odb`, created using the **abaqus emloads** utility, and which has mapped quantities from the results of the time-harmonic eddy current analysis, stored in `electromagnetic.odb`:

```
*HEADING
…
*ELEMENT, TYPE=DC3D8
(Choose the heat transfer continuum element type)
…
*STEP
*HEAT TRANSFER, STEADY STATE
…
Apply heat transfer loads and boundary conditions
…
*CFLUX, FILE=pointflux.odb
Read in all nodal heat flux values from the output database and apply as concentrated nodal fluxes
…
*END STEP
```

Co-Simulation

In this section:

- *About Co-Simulation*
- *Preparing an Abaqus Analysis for Co-Simulation*
- *Co-Simulation between Abaqus Solvers*
- *System-Level Modeling between Logical and Physical Interactions*

About Co-Simulation

Overview

The co-simulation solution technique uses a segregated solution approach in which your multiphysics or multiscale problem is divided into two or more subsystems, where each subsystem is solved independently with solution data exchanged during the course of the analysis. An Abaqus analysis can be coupled to another Abaqus analysis or to another analysis program for solving multiphysics and multiscale simulations.

Abaqus provides built-in procedures to solve multiphysics simulations as described in *Multiphysics Analyses*. For multiphysics problems for which Abaqus does not provide a built-in solution procedure or where the solution procedure is limited in functionality, you can use the co-simulation solution technique to couple Abaqus with another analysis program; for example, fluid-structure interaction (FSI) simulation in conjunction with computational fluid dynamics (CFD) analysis programs.

Co-simulation between Abaqus/Standard and Abaqus/Explicit illustrates a multiple domain analysis approach, where each Abaqus analysis operates on a complementary section of the model domain where it is expected to provide the more computationally efficient solution. For example, Abaqus/Standard provides a more efficient solution for light and stiff components, while Abaqus/Explicit is more efficient for solving complex contact interactions.

Features of the Abaqus Co-Simulation Solution Technique

The Abaqus co-simulation technique:

- can be used to solve complex fluid-structure interactions by coupling Abaqus with the SIMULIA Navier-Stokes solver (available in the Fluid Scenario Creationapp on the **3D**EXPERIENCE platform) or with a third-party CFD solver;
- can be used to solve problems involving electromagnetic-thermal or electromagnetic-mechanical interactions by coupling Abaqus with an electromagnetic analysis program, including electromagnetic analysis procedures in Abaqus/Standard, CST Studio Suite, or a third-party electromagnetic solver;
- can be used to solve general multiphysics simulations by coupling Abaqus with third-party and in-house analysis programs;
- can be used to solve complex multidomain analyses more effectively by coupling Abaqus/Standard to Abaqus/Explicit and Abaqus to Simpack;
- can be used for system-level modeling between logical and physical components; for example, coupling Abaqus with Dymola or any Functional Mockup Unit (FMU) using the FMI 2.0 standard;
- uses the SIMULIA Co-Simulation Services to coordinate the coupled simulation;
- is intended for advanced users with in-depth knowledge of Abaqus, the other analysis program, and coupled solution methods;
- can be used with Abaqus models having linear or nonlinear structural response; and
- supports steady-state and transient procedures and time-harmonic procedures for electromagnetics.

SIMULIA Co-Simulation Services

The SIMULIA Co-Simulation Services (CSS) is a software component in charge of coordinating coupled simulations. Coupled simulations include sequential simulations (see *General Capability for Importing External Fields*), submodeling (see *Node-Based Submodeling*), and co-simulation workflows. The SIMULIA Co-Simulation

Services includes the CSS Director, a separate executable in charge of controlling the coupled simulation, and software services embedded into the solver applications for communication, sequencing, and field mapping. *Coupled Simulation Interface* discusses the SIMULIA Co-Simulation Services, providing a general overview of the services, and technical information on the CSS-API and CSS Configuration File. The technical information is intended for users who want to couple their application with Abaqus.

Interaction between Domains Modeled with Different Analysis Programs

In a co-simulation the interaction between the numerical domains is through a common physical interface region over which data are exchanged in a synchronized manner between Abaqus and the coupled analysis programs. When coupling numerical domains to solve a multiphysics/multiscale problem, you must consider whether the numerical domains overlap or abut each other in order to define the domain coupling type. Domain coupling types include point coupling, surface coupling, and volume coupling (see *Domain Coupling for Multiphysics and Multiscale Simulations*).

One domain can affect the response of another domain through one or more of the following interactions:

- the constitutive behavior, such as the yield stress defined as a function of temperature or stress defined as a function of other solution fields, such as thermal strains or the piezoelectric effect;
- surface tractions/fluxes, such as a fluid exerting pressure on a structure;
- body forces/fluxes, such as Joule heating due to electrical current flow in a coupled thermal-electrical simulation;
- contact forces, such as the forces due to contact between a vehicle and an occupant/pedestrian modeled as separate domains;
- kinematics, such as fluid in contact with a compliant structure where the interface motion affects the fluid flow; and
- discrete coupling, such as sensor and actuation information.

Co-Simulation Workflows with Abaqus

Co-simulation extends the problems that can be solved in Abaqus. It can combine solvers with different numerical methods to solve complex multiphysics and multiscale problems that otherwise cannot be solved using a monolithic solution approach. Co-simulation can provide a higher-fidelity solution by adding more complex interactions, and it can reduce computational time by coupling lower-order models with 3D physical simulations to analyze complex systems. Some common Abaqus multiphysics and multiscale workflows are illustrated below.

Fluid-Structure Interaction

You can solve complex fluid-structure interaction (FSI) problems by coupling Abaqus/Standard or Abaqus/Explicit to a computational fluid dynamics (CFD) analysis program. Abaqus solves the structural domain, and the CFD analysis program solves the fluid domain. Abaqus can be coupled with the SIMULIA Navier-Stokes solver (available in the Fluid Scenario Creationapp on the **3D**EXPERIENCE platform) or with several third-party CFD analysis programs.

For a complete list of qualified partner products, see the *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* page at *https://www.3ds.com/products-services/simulia/*.

Conjugate Heat Transfer

You can solve conjugate heat transfer problems involving fluids and structures by coupling Abaqus/Standard to a computational fluid dynamics (CFD) analysis program. Abaqus/Standard models heat transfer within the solid (see *Uncoupled Heat Transfer Analysis* and *Fully Coupled Thermal-Stress Analysis*), and the CFD analysis

program solves the energy equation for the fluid flow surrounding the structure. Abaqus/Standard can be coupled with the SIMULIA Navier-Stokes solver (available in the Fluid Scenario Creationapp on the **3D**EXPERIENCE platform) or several third-party CFD analysis programs.

For a complete list of qualified partner products, see the *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* page at *https://www.3ds.com/products-services/simulia/*.

Electromagnetic-Thermal or Electromagnetic-Mechanical Coupling

Applications such as induction heating require interaction between electromagnetic and thermal fields. You can solve this class of problems by coupling Abaqus/Standard with itself, where one analysis solves for the fields in the electromagnetic domain, while the other solves for the fields in the thermal domain. Alternatively, you can couple Abaqus/Standard with CST Studio Suite or with several third-party electromagnetic analysis programs.

For detailed information on coupling Abaqus/Standard to Abaqus/Standard, see *Preparing an Abaqus Analysis for Co-Simulation* and *Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation*. For a complete list of qualified partner products, see the *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* page at *https://www.3ds.com/products-services/simulia/*.

System-Level Modeling via Logical-Physical Interaction

System-level modeling refers to modeling of systems that can include both physical (structural, thermal, acoustics, etc.) and logical components modeled via a Function Mock-up Unit (FMU) using the Functional Mock-up Interface (FMI). For more information about FMU and FMI, see *http://www.fmi-standard.org*.

The distinction between the logical and physical modeling abstractions is as follows:

- Logical modeling refers to a large class of modeling abstractions often encountered in the engineering practice. Generally speaking, you can designate a part of a system as using a logical modeling abstraction when most (if not all) of the geometry of the part is removed. Examples include electronic control modules, electric motors, and pneumatic or hydraulic subsystems, which in many cases can be modeled from a functional perspective without attempting to model the flow of electrons, the variation of magnetic fluxes, or the air/fluid type of flow in ducts and pipes. Dymola and other third-party products offer a variety of logical modeling options by generating FMUs that can be consumed in Abaqus co-simulation.
- Physical modeling is the complementary modeling abstraction to logical modeling. Abaqus uses a physical modeling abstraction most of the time; as elements deform, they know precisely about their geometry, thus trying to mimic the real world at a fine-grain level.

In many engineering systems the interaction between logical and physical components is paramount, and you cannot fully analyze one without the other. Co-simulation using Abaqus and FMUs provides the capability to analyze this type of system.

Consider the example of a rolling mill: the incoming slab, which might not have a constant thickness, can be modeled in Abaqus as being deformed by the rolling cylinders. Because of the nonconstant incoming thickness, a pressure that adapts as a function of deformation needs to be exerted on the cylinders to compensate such that the exit thickness is as constant as possible. Abaqus sensors can export the information about the mechanical status of the system to FMUs, which in turn could use this information to model the necessary compensators to calculate the needed actuation load at any given time. Abaqus can import the actuation load and apply it to the cylinders.

For detailed information on coupling Abaqus/Standard to FMUs, see *System-Level Modeling between Logical and Physical Interactions*. For a complete list of qualified partner products, see the *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* page at *https://www.3ds.com/products-services/simulia/*.

Structure-to-Structure Multidomain Simulations

In certain cases you can realize significant computational cost savings by partitioning a model and combining the Abaqus/Standard and Abaqus/Explicit solutions, such as

- when the simulation is principally a candidate for Abaqus/Explicit, but where certain parts of the model can be idealized using substructures in Abaqus/Standard, or
- when the simulation is principally a candidate for Abaqus/Standard, but where complex contact conditions would be handled more effectively by Abaqus/Explicit.

For an example of Abaqus/Standard to Abaqus/Explicit co-simulation, refer to *Dynamic impact of a scooter with a bump*. For detailed information on coupling Abaqus/Standard and Abaqus/Explicit, see *Preparing an Abaqus Analysis for Co-Simulation* and *Structural-to-Structural Co-Simulation*.

In certain cases, you might want to consider a Multibody Dynamics Solver to obtain a cost-effective solution and to introduce a nonlinear part modeled with Abaqus. You can accomplish this by performing a co-simulation between Simpack and Abaqus.

Both the Abaqus/Standard to Abaqus/Explicit and Abaqus to Simpack solutions use highly specialized coupling algorithm to provide robust coupling in a numerically cost-effective manner.

Coupling Using the MpCCI Interface

MpCCI, the multiphysics code coupling interface developed and distributed by the Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI), provides an open system approach for general multidisciplinary simulations between Abaqus and any third-party analysis program that supports MpCCI. MpCCI provides a scalable communication infrastructure and mapping algorithms for multiple physics domains. In a co-simulation using MpCCI, Abaqus communicates with the MpCCI coupling server to exchange fields with the third-party analysis program while each analysis advances its simulation time.

Coupling through MpCCI can occur between Abaqus and any third-party analysis program that supports the MpCCI interface. This includes in-house codes that have the MpCCI adapter embedded. The Fraunhofer-Institute for Algorithms and Scientific Computing actively supports and qualifies a link between Abaqus and MpCCI, which provides indirect coupling to FLUENT and OpenFoam for fluid-structure interaction. For further information on coupling using the MpCCI interface, contact *https://www.mpcci.de/*.

Additional Applications

There are many other applications for which co-simulation can be employed with partner products. For example, vehicle ride comfort and durability simulation using FTire from *https://www.cosin.eu/* and occupant and pedestrian safety using MADYMO from *https://tass.plm.automation.siemens.com/*. For a complete list of qualified partner products, see the *https://www.3ds.com/support/hardware-and-software/simulia-system-information/compatibility/co-simulation/* page at *https://www.3ds.com/products-services/simulia/*.

Strength of Physics Coupling and Coupling Algorithms

You typically apply co-simulation techniques to problems where the most complex physics occurs within domains that are handled exclusively within an analysis. Due to the comparative numerical simplicity of the numerical techniques applied at the co-simulation interface, the physics controlling the interaction at the interface of the separate analysis domains (the strength of the physics coupling) might lead to stability and accuracy concerns.

Figure 1 illustrates the coupling strength with an analogy in the frequency domain. Consider a lumped parameter dynamic system with a coupling impedance directly related to a response frequency ω. In a staggered solution approach each domain is solved by temporarily ignoring the coupling terms represented by the gray spring and dashpot in *Figure 1*.

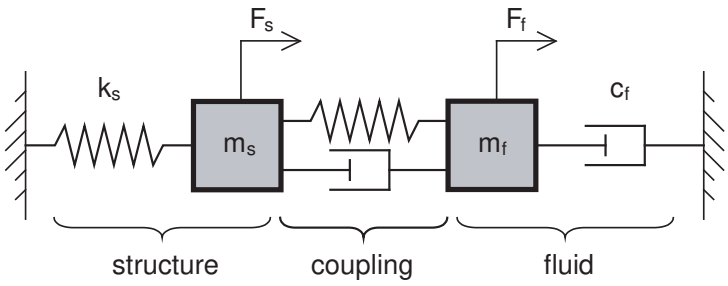


Figure 1: Mechanical impedance analogy.

When the response frequency and coupling impedance are low, a staggered approach likely provides adequate solution accuracy and performance. However, when the response frequency is high (such that the coupling impedance is relatively large compared to the structure or fluid), you might encounter solution stability issues with the staggered approach.

Analysis domains are coupled in a staggered approach using either a globally explicit manner or an implicit iterative manner; that is, the equations for each domain are solved separately, and loads and boundary conditions are exchanged at the common interface. The SIMULIA Co-Simulation Services supports different coupling schemes that can deliver robust and cost effective solutions to problems exhibiting different degrees of physics coupling strength.

In cases where the coupling is sufficiently weak, the coupling might be required only in one direction (such as when an electromagnetic force field contributes to the structural response, but a reverse coupling provides no significant impact on the electromagnetic field). A sequential solution approach is the most cost effective solution technique to solve these problems, although you can also run both solvers concurrently and exchange solution data in one direction only.

In an explicit staggered approach (such as the Gauss-Seidel coupling scheme), fields are exchanged only once per coupling step. This coupling strategy is applicable to problems that exhibit weak to moderate physics coupling (for example, aeroelasticity problems where you have air interacting with a relatively stiff structure). The explicit staggered approach is numerically cost effective, but it requires the use of a smaller coupling step size to obtain a stable and accurate solution.

In an implicit iterative approach, the fields are exchanged multiple times per coupling step until an overall equilibrium is achieved prior to advancing to the next coupling step. Implicit coupling is computationally more expensive per coupling step; however, in general, a larger coupling step size can be employed. Implicit schemes are appropriate for problems exhibiting moderate to strong physics coupling (for example, blood flow through an artery were the fluid density ratio between the fluid and solid is near unity, and the fluid is incompressible); however, this approach is limited to solvers that employ implicit time integration.

Accelerators can be employed with the implicit iterative approach to enhance the coupling by enlarging the convergence radius from a stability point of view. The SIMULIA Co-Simulation Services supports two types of accelerator methods (relaxation technique and Quasi-Newton methods that approximate an inverse Jacobian at the interface) and can deliver close to an exact inverse Jacobian if sufficient past residual information is available.

The strength of the physics coupling can generally be greater when coupling two mechanical systems; for example, Abaqus/Standard to Abaqus/Explicit or Abaqus to Simpack. The SIMULIA Co-Simulation Services supports coupling algorithms through communication of “right-hand-side” and “left-hand-side” terms to provide robust interface solutions across a wide range of problem parameters, which supports both implicit and explicit solvers.

Dual Mass-Spring Systems illustrate the stability and accuracy issues associated with the partitioned solution approach for a simple dual mass-spring system. The example, compares different interface treatment methods availble to deliver robust solutions.

References

For the latest support information and tips on running co-simulations with third-party analysis programs, see the Dassault Systèmes Knowledge Base at *https://support.3ds.com/knowledge-base/*.

Preparing an Abaqus Analysis for Co-Simulation

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Co-Simulation*
- *Co-Simulation between Abaqus Solvers*
- *Structural-to-Structural Co-Simulation*
- *Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation*

Overview

Preparing an Abaqus analysis for co-simulation involves the following:

- identifying an Abaqus analysis step for co-simulation analysis;
- identifying the co-simulation interface regions in the Abaqus model; and
- identifying the fields exchanged during the co-simulation.

This section provides an overview of preparing an Abaqus analysis for a co-simulation using the SIMULIA Co-Simulation Services. The discussion in this section is general and might not apply to every product pairing.*Co-Simulation between Abaqus Solvers* provides setup, execution, and limitation details for co-simulation between Abaqus solvers. For co-simulation between Abaqus and third-party analysis programs, consult the appropriate User's Guide.

Identifying an Abaqus Step for Co-Simulation Analysis

The co-simulation event need not begin at the start of the first step in an Abaqus analysis. However, it does need to start with the beginning of an analysis step and end within that analysis step. Hence, you need to define the step durations in Abaqus such that the start of the co-simulation event falls at the beginning of an Abaqus analysis step and to define that particular step so that the co-simulation event ends by the end of that step. Regular loads and boundary conditions for the Abaqus model are specified as usual.

Communication with the coupled analysis is initiated as the co-simulation event begins and is terminated when the co-simulation event time is reached. Abaqus might terminate the co-simulation event when the end of the analysis step is reached before the co-simulation event time or when the analysis cannot proceed any further; for example, because of convergence problems. In such a case, a warning message is issued to all clients, and the co-simulation is terminated.

Co-simulation is supported by the following Abaqus procedures:

- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Explicit Dynamic Analysis*
- *Uncoupled Heat Transfer Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Piezoelectric Analysis*

- *Coupled Thermal-Electrical Analysis*
- *Eddy Current Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*

Input File Usage: Use the following option within a step definition to indicate the beginning of a co-simulation event:

**CO-SIMULATION*,PROGRAM=MULTIPHYSICS

Identifying the Co-Simulation Interface Region

Interaction between two Abaqus models or between an Abaqus model and a third-party analysis model takes place through a common interface region referred to as the co-simulation interface region. The co-simulation interface region can be a set of discrete points, a surface region, or a volume region (see *Domain Coupling for Multiphysics and Multiscale Simulations*). You must be consistent in your interface region definition; if you define a surface co-simulation region in one analysis, you must define a surface co-simulation region in the other analysis. In addition, these co-simulation regions need to be colocated and have the same region boundaries. The regions should not overlap unless different physics interactions are transferred.

Interacting through Discrete Points

Interaction can occur through a set of discrete points where only nodal position information without element topology information (for example, tributary area) defines the co-simulation interface region. In this case the spatial mapping is limited to point-to-point mapping, and you must ensure that there are matching nodes between the models. Interaction through discrete points is commonly employed when coupling Abaqus with a one-dimensional model.

In Abaqus you can use a node set or a node-based surface to define a co-simulation interface region consisting of discrete points.

Input File Usage: Use the following option to define a node set as a co-simulation region in an Abaqus model:

**CO-SIMULATION REGION*, TYPE=NODE
nodeset_A

Use the following options to define a node-based surface as a co-simulation region in an Abaqus model:

**SURFACE*, TYPE=NODE
nodeset_A
**CO-SIMULATION REGION*, TYPE=SURFACE
node-based surface name

Interacting through a Surface

Interaction between distinct domains occurs through a common interface surface. For example, when a fluid interacts with a solid without penetrating it, the fluid-solid interface is defined through a common surface. In this case both nodal position and element topology information define the co-simulation interface, and appropriate spatial mapping between dissimilar surface meshes is performed to conservatively map fields.

Input File Usage: Use the following option to define an element-based surface as a co-simulation region in an Abaqus model:

**CO-SIMULATION REGION*, TYPE=SURFACE (default)
element-based surface name

Interacting through a Volume

Interaction between overlapping domains occurs through a volume intersecting both numerical domains. Typical examples of volumetric interaction include coupling Abaqus with an electromagnetic solver and Abaqus with a reservoir solver. In this case both nodal position and element topology information define the co-simulation region, and appropriate spatial mapping between dissimilar volume meshes is performed conservatively.

The interface region is defined by an element set.

Input File Usage: Use the following option to define a volume as a co-simulation region in an Abaqus model:

**CO-SIMULATION REGION*, TYPE=VOLUME
elset_A

Identifying the Fields Exchanged across a Co-Simulation Interface

The coupling of the domain models can be through loads and/or boundary conditions prescribed at the co-simulation interface. In addition, mass, rotary inertia, and heat capacitance terms can also be exchanged. Based on the physics and the interaction type and its enforcement, you must specify the fields that are imported and/or exported in an Abaqus analysis during the co-simulation.

The co-simulation interface can consist of a group of discrete points (nodes), a surface region, or a volume region. Not all fields can be exchanged across all region types. For example, pressure (that is, force per unit area) is a surface quantity, and it cannot be transferred at discrete points or over a volume.

This section provides a general overview of all fields available in Abaqus. For detailed information on the fields exchanged between two Abaqus solvers, see *Structural-to-Structural Co-Simulation*. For detailed information on fields exchanged by Abaqus and a third-party analysis program, see the respective User's Guides.

Input File Usage: Use the following option to import field data over a region into Abaqus:

**CO-SIMULATION REGION*, IMPORT
region_A, *import_field_1*
region_A, *import_field_2*

Use the following option to export data from Abaqus:

**CO-SIMULATION REGION*, EXPORT
region_A, *export_field_1*
region_A, *export_field_2*

Procedures Involving Mechanical Degrees of Freedom

Table 1 lists the fields that can be exchanged for procedures supporting mechanical degrees of freedom (degrees of freedom 1–6), their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 1: Exchanging fields for procedures supporting mechanical degrees of freedom.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
UT or U	Displacement	P, S, V	S, E	S, E	L
VT or V	Velocity (transient procedures)	P, S, V	S, E	S, E	LT^{-1}
AT or A	Acceleration (transient procedures)	P, S, V	S	S, E	LT^{-2}
UR	Rotations	P, S	S, E	S, E	radians
VR	Angular velocity (transient procedures)	P, S	S, E	S, E	radians T^{-1}
AR	Angular acceleration (transient procedures)	P, S	S	S, E	radians T^{-2}
COORD	Current coordinates	P, S, V		S, E	L
CF	Concentrated forces	P, S, V	S, E	S, E	F
CM	Concentrated moments	P, S	S, E	S, E	FL
P	Pressure normal to element surface	S	S		FL^{-2}
TRVEC	Traction vector	S	S		FL^{-2}
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

The following procedures support co-simulation using mechanical degrees of freedom:

- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Explicit Dynamic Analysis*
- *Fully Coupled Thermal-Stress Analysis*
- *Piezoelectric Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*

Displacements

Displacements (field IDUT or U) for the translational degrees of freedom can be exported by Abaqus/Standard and Abaqus/Explicit. Displacements can be imported by Abaqus/Standard and Abaqus/Explicit. When imported, displacements are ramped from the values of the previous exchange time point to those of the next target time point. The displacements are in the global coordinate system.

Displacements are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Displacements can be viewed in the Visualization module of Abaqus/CAE.

Velocity and Acceleration

Velocity (field IDVT or V) and acceleration (field IDAT or A) for the translational degrees of freedom can be imported and exported by Abaqus/Standard for transient procedures and by Abaqus/Explicit. Velocity and acceleration are in the global coordinate system.

Velocity and acceleration are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Velocity and acceleration can be viewed in the Visualization module of Abaqus/CAE.

Rotations

Rotations (field IDUR) can be imported and exported by Abaqus/Standard and Abaqus/Explicit. In an implicit dynamic analysis, rotational velocity and rotational acceleration must be imported when importing rotations. Rotations are in radians and in the global coordinate system.

Rotations are available for points and surface regions.

Rotations can be viewed in the Visualization module of Abaqus/CAE.

Rotational Velocity and Rotational Acceleration

Rotational velocity (field IDVR) and rotational acceleration (field IDAR) can be imported and exported by Abaqus/Standard for transient procedures and by Abaqus/Explicit. Rotational velocity and acceleration are in radians per time and time squared, respectively, and are in the global coordinate system.

Rotational velocity and rotational acceleration are available for points and surface regions.

Rotational velocity and acceleration can be viewed in the Visualization module of Abaqus/CAE.

Current Coordinates

Current nodal coordinates (field IDCOORD) can be exported by Abaqus/Standard and Abaqus/Explicit. The coordinates are the current coordinates of the deformed structure whether small- or large-displacement analysis is performed. In general, it is preferred to export displacements (field IDUT or U) rather than current coordinates when results are mapped between dissimilar interface regions. In cases where the partner client does not retain the original coordinates, it might be necessary to send current coordinate values rather than displacements.

Current coordinates are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated Forces

Concentrated forces (field IDCF), if imported, are ramped from the values of the previous exchange time point. The concentrated forces are in the global coordinate system.

When exporting concentrated forces, Abaqus/Standard transfers reaction forces at interface nodes that have prescribed displacements. The reaction forces are exported in the global coordinate system.

Concentrated forces are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated normal forces can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CF and for Abaqus/Explicit simulation by requesting output variable FEXT.

Concentrated Moments

Concentrated moments (field IDCM), if imported, are ramped from the values of the previous exchange time point to those of the next target time point. The concentrated moments are in the global coordinate system.

Concentrated moments are available for points and surface regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated moments can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CM and for an Abaqus/Explicit simulation by requesting output variable MEXT.

Normal Pressure

Normal pressure (field IDP), supported for import by Abaqus/Standard, is the traction normal component to the surface. Pressure values are ramped from the values of the previous exchange time point to those of the next target time point when imported into Abaqus/Standard. In most cases it is preferred to import concentrated forces (field IDCF) because these contain both the normal and shear traction components. For shell and membrane-like structures, it might be preferable to import pressures rather than concentrated forces.

Normal pressure can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable P.

Traction Vector

Traction vector (field IDTRVEC), supported for import by Abaqus/Standard, is a general traction vector acting on a surface. Traction values are ramped from the values of the previous exchange time point to those of the next target time point when imported into Abaqus/Standard. For shell and membrane-like structures, it might be preferable to import traction rather than concentrated forces.

Normal pressure can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable P.

Procedures Involving Thermal Degrees of Freedom

Table 2 lists the thermal fields available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 2: Exchanging fields for procedures supporting thermal degrees of freedom.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
NT	Nodal temperature	P, S, V	S, E	S, E	θ
CFL	Concentrated heat flux at a node	P, S, V	S, E		JT^{-1}
HFL	Heat flux normal to element surface	S	S		$JT^{-1}L^{-2}$
FILMCOEF	Film properties	S	S		$JT^{-1}L^{-2}\theta^{-1}$
SINKTEMP	Sink temperature	S	S		θ
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

The following procedures support co-simulation using thermal degrees of freedom:

- *Uncoupled Heat Transfer Analysis*

- *Fully Coupled Thermal-Stress Analysis*
- *Coupled Thermal-Electrical Analysis*

Nodal Temperature

Nodal temperature (field IDNT) can be imported and exported by Abaqus/Standard and Abaqus/Explicit. Temperature values are ramped from the values of the previous exchange time point to those of the next target time point.

Temperature values can be exchanged either on the top surface (SPOS) or the bottom surface (SNEG) of structural elements. Temperatures cannot be exchanged on double-sided surfaces. When exchanging temperatures on both the top and bottom faces, define two different regions; one to exchange temperature on the top face and the other to exchange temperature on the bottom face. Temperatures cannot be exchanged through the volume of a shell element.

Nodal temperature values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable NT.

Heat Flux

Use concentrated heat flux (field IDCFL) for heat entering at a node in Abaqus/Standard and Abaqus/Explicit. Concentrated heat flux is available for points, surface regions, and volume regions.

Heat flux values can be exchanged either on the top surface (SPOS) or the bottom surface (SNEG) of structural elements. Heat flux cannot be exchanged on double-sided surfaces. When exchanging heat flux on both the top and bottom faces, define two different regions; one to exchange heat flux on the top face and the other to exchange heat flux on the bottom face. Heat flux cannot be exchanged through the volume of shell elements.

Concentrated heat flux values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CFL.

Use surface heat flux (field IDHFL) for a distributed heat flux entering the surface in Abaqus/Standard. Distributed heat flux is available only for surface regions.

Surface heat flux can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable FLUXS.

Film Properties

Use surface film properties (field IDFILMCOEF and SINKTEMP) to model convection governed by

$$q = -h\left(\theta_{wall} - \theta_{fluid}\right),$$

where q is the heat flux entering the surface, h is a film coefficient, θ_{wall} is the wall temperature, and θ_{fluid} is the fluid or sink temperature.

Both the film coefficient and fluid temperature need to be imported into Abaqus/Standard and are kept constant over the subsequent exchange interval. When the fluid and wall temperatures coincide, an arbitrary small value for the heat transfer coefficient is passed into Abaqus. To obtain reasonable heat flux for the first exchange interval, you need to ensure that the wall temperatures are initialized properly in Abaqus and that you provide a good estimate for the initial fluid temperature.

Film properties are available only for surface regions in Abaqus/Standard.

Procedures Involving Pore Fluid Pressure

Table 3 lists additional fields that can be exchanged for a coupled pore fluid diffusion/stress analysis, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 3: Exchanging fields for a coupled pore fluid diffusion/stress analysis.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
POR	Pore fluid pressure at a node	P, S, V	S	S	FL^{-2}
CFF	Concentrated fluid flow at a node	P, S, V	S		L^3T^{-1}
RVF	Reaction fluid volume flux due to prescribed pressure	P, S, V		S	L^3T^{-1}
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

The following procedure involving pore fluid pressure supports co-simulation:

- *Coupled Pore Fluid Diffusion and Stress Analysis*

Pore Pressure

Nodal pore pressure (field IDPOR) can be imported and exported by Abaqus/Standard for points, surface regions, and volume regions.

Nodal pore pressure values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable POR.

Concentrated Fluid Flow

Fluid flow (field IDCFF) defines the seepage flow at a node. Concentrated fluid flow can be imported by Abaqus/Standard for points, surface regions, and volume regions.

Concentrated fluid flow values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CFF.

Reaction Fluid Volume Flow

Reaction fluid volume flux (field IDRVF) defines the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pore pressure. Reaction fluid volume flux can be exported by Abaqus/Standard for points, surface regions, and volume regions.

Procedures Involving Electromagnetic Response

Table 4 lists additional fields that can be exchanged for an electromagnetic analysis, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 4: Exchanging fields for a electromagnetic analysis.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
EMJH	Joule heating flux due to flow of current	V		S	$JT^{-1}L^{-3}$
EMBF	Magnetic body force intensity vector due to flow of induced current	V		S	$FT^{-1}L^{-3}$
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

The following procedure involving electromagnetics supports co-simulation:

- *Eddy Current Analysis*

Joule Heating Flux

The Joule heating flux (field IDEMJH) can be exported by Abaqus/Standard for volume regions. It can be imported in a downstream heat transfer analysis as concentrated nodal heat flux (field IDCFL).

Values for the Joule heating flux can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable EMJH.

Magnetic Body Force Intensity Vector

The magnetic body force intensity vector (field IDEMBF) can be exported by Abaqus/Standard for volume regions. It can be imported in a downstream stress analysis as concentrated force (field IDCF).

Magnetic body force intensity vector values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable EMBF.

Temperature and Independent Field Variables

Field variables are time-dependent, predefined fields that exist over the spatial domain of the model (see *Predefined Fields*). Field variables in conjunction with the co-simulation technique extend the possibilities of multiphysics by allowing material point dependencies on an external field defined by another application.

Field variables must be numbered consecutively starting with one. Field variables can be defined:

- by entering the data directly,
- by reading an Abaqus results file or output database file,
- in an Abaqus/Standard user subroutine, and
- through the co-simulation interface.

If field variables are defined by multiple methods, Abaqus processes them in the order defined above. Care needs be taken when field variables are used with structural elements, such as membranes and shells. In this case only the top or bottom face forming the interface region receives a value.

Table 5 lists the temperature and independent field variables available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 5: Exchanging temperature and independent field variables.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
NT	Temperature as field variable specified at nodes but interpolated to integration points	V	S		θ
FV1	Field variable 1	V	S		
FV2	Field variable 2	V	S		
FV3	Field variable 3	V	S		
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

The following Abaqus/Standard procedures support import of temperature and independent field variables:

- *Static Stress Analysis*
- *Quasi-Static Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Fully Coupled Thermal-Stress Analysis*
- *Piezoelectric Analysis*
- *Eddy Current Analysis*
- *Coupled Pore Fluid Diffusion and Stress Analysis*

Temperature as a Field and Field Variables

Temperature (field IDNT) can be imported for procedures that allow material properties to be defined as a function of an external temperature field. When imported, temperature values are ramped from the values of the previous exchange time point to those of the next target time point.

Independent field variables (field IDs FV1, FV2, and FV3) can be imported by Abaqus/Standard, allowing material properties to be defined as a function of the external fields. When imported, independent field variable values are ramped from the values of the previous exchange time point to those of the next target time point.

Temperature as a field variable can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation. You might notice differences when you compare the source and target fields. There are two possible causes for these differences:

- Field variables and temperature as a field variable are mapped accurately and imported at nodes. However, when saved to the output database, these quantities are interpolated to element integration points. When contouring, some averaging can occur depending on the contouring parameters selected.
- For fully integrated first-order elements, a selective reduced-integration technique is employed (see *Solid isoparametric quadrilaterals and hexahedra*). In this case, field variables and temperature as a field variable are averaged at the element center and saved to the output database.

Miscellaneous Fields

Table 6 lists miscellaneous fields available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 6: Exchanging miscellaneous fields.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
MASS or LUMPEDMASS	Mass	P, S	S, E	S, E	M
RI	Rotary inertia	P, S	S	E	ML^2
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit)					

Lumped Mass

Lumped mass values (field IDMASS or LUMPEDMASS) at nodes can be exported and imported by Abaqus/Standard and Abaqus/Explicit.

Lumped mass is available for points and surface regions.

Rotary Inertia

Nodal (lumped) rotary inertia (field IDRI) can be imported by Abaqus/Standard and exported by Abaqus/Explicit over points or surface regions for models using structural elements.

Defining the Coupling Algorithm

The coupling algorithm defines the sequence of exchanges between the analysis programs and any special interface calculations that are performed on the interface. When deciding on the coupling algorithm, you should consider solution stability issues as well as the utilization impact on your computing resources. The SIMULIA Co-Simulation Services overs a wide choice of coupling algorithms (see *Strength of Physics Coupling and Coupling Algorithms*) including explicit schemes, implicit iterative schemes with a choice of accelerators, and specialized coupling algorithms (for Abaqus/Standard to Abaqus/Explicit or Abaqus to Simpack co-simulation). The choice of the coupling scheme also depends on the solvers participating in the co-simulation; for example, you cannot use implicit iterative coupling if one of the solvers uses explicit time integration. Consideration should also include the numerical cost associated with interface computations; for example, the enhanced sub-cycling method can become excessively costly for large interfaces, and the Quasi-Newton accelerators might be too expensive when coupling large volumetric domains.

The coupling algorithm and its associated parameters are defined in the SIMULIA Co-Simulation Services configuration file. For guidance in selecting an appropriate coupling algorithm for co-simulation between Abaqus solvers, see *Co-Simulation between Abaqus Solvers*; for co-simulation between Abaqus and third-party analysis programs, consult the appropriate user's guide.

Explicit Coupling Schemes

In an explicit staggered approach fields are exchanged once per coupling step, and no convergence check is performed before the solvers advance to the next coupling step. You should employ the explicit staggered approach when one of the solvers uses explicit time integration or for weak to moderate physics coupling (for example, aeroelasticity problems where you have air interacting with a relatively stiff structure). The explicit staggered approach is numerically cost effective, but it requires the use of a smaller coupling step size to obtain a stable and accurate solution.

The solvers can be executed in parallel (commonly referred to as the Jacobi scheme) or sequentially (commonly referred to as the Gauss-Seidel scheme). In the Jacobi scheme, both solvers execute at the same time and both solutions lag by one coupling step. The Jacobi scheme might make more efficient use of computing resources;

however, it is considered less stable than the Gauss-Seidel scheme. In the Gauss-Seidel scheme, the simulations are executed in sequential order; and one analysis leads while the other analysis lags the co-simulation.

Implicit Iterative Coupling Schemes

In an implicit iterative coupling scheme, the simulations are executed in sequential order. When commencing a coupling step, one analysis leads while the other analysis lags the co-simulation. However, multiple exchanges (referred to coupling iterations) are performed at the target point until global convergence (convergence between all solvers) is achieved, prior to commencing the next coupling step. The additional coupling iterations remove the lag between the solutions that exists for explicit coupling.

Implicit coupling is numerically more costly because multiple coupling iterations are performed; however, in general, a larger coupling step size can be used to obtain a stable solution. You should use implicit coupling for strong physics coupling if all solvers use implicit time integration.

Accelerators for the implicit iterative schemes enhance the coupling by enlarging the convergence radius from a stability point of view. The SIMULIA Co-Simulation Services supports two types of accelerator methods:

- Relaxation is a numerical technique employed commonly to improve the stability of a computation. Using a relaxation factor less than one improves the stability at the expense of slower convergence (more coupling iterations). Relaxation works well for moderate-strength physics coupling.
- The Quasi-Newton methods approximate an inverse Jacobian at the interface and can deliver close to an exact inverse Jacobian if sufficient past residual information is available. The Quasi-Newton method incurs an interface cost but can significantly reduce the number of coupling iterations. It is best suited for strongly coupled physics.

Specialized Coupling Schemes

The SIMULIA Co-Simulation Services supports specialized coupling schemes through communication of “right-hand-side” and “left-hand-side” terms to provide robust interface solutions, which support coupling of solvers that use different time integration schemes (for example, explicit versus implicit time integrators). These coupling scheme provide robust and cost effective solutions and are employed for coupling Abaqus/Standard to Abaqus/Explicit or Abaqus to Simpack.

Rendezvousing Schemes

Different types of analyses have different time integration requirements that influence or dictate the frequency of interaction between the analyses in a co-simulation to obtain an accurate and robust solution. For example, consider the difference in increment size between an implicit and an explicit dynamic analysis. In addition, Abaqus/Standard can adjust the increment sizes automatically to obtain an economical and accurate solution for transient problems (see *Incrementation*). For example, consider a transient heat transfer analysis modeling a diffusive process. Here, the analysis might use small time increments at the beginning of the analysis where there is a high gradient in the solution and large time increments toward the end of the analysis when steady state is reached.

The coupling step is the period between two consecutive exchanges. Consequently, it defines the frequency of exchange between the analyses in a co-simulation and directly influences the stability and accuracy of the coupled solution. The coupling step size is established at the beginning of each coupling step and is used to compute the next target time (the time when the next synchronized exchange of field data occurs). The SIMULIA Co-Simulation Services determines the next coupling step's size based on the analyses' preferred increment size and the user-selected negotiation scheme. Abaqus always uses the increment size suggested by its automatic time incrementation as the preferred increment size for determining the coupling step size.

Typically, Abaqus might perform several increments (referred to as “subcycling”) during the coupling step. During subcycling, Abaqus ramps the loads and boundary conditions from the values at the end of the previous coupling step to the values at the target time. Subcycling allows Abaqus to use its own time incrementation to

reach the target coupling time; specifically, it allows Abaqus to cut back the increment size if there are nonlinear events that require the increment size to be reduced. However, subcycling might come at the expense of solution robustness and accuracy.

In certain cases you can force Abaqus to use a time increment size equal to the coupling step size (referred to as lockstep). This approach allows both solvers to use the same time incrementation and avoid interpolation of quantities during the coupling step. It is best from a solution robustness and accuracy point of view. When proceeding in this “lockstep” manner, Abaqus is not able to reduce the time increment to resolve nonlinear events and, consequently, ends the simulation in cases where the nonlinear events require that the increment size be reduced. In most cases it is best to subcycle and take a penalty in terms of solution robustness and accuracy. Subcycling allows Abaqus to use its own time incrementation to reach the target coupling time; specifically, it allows Abaqus to cut back the increment size if there are nonlinear events that require the increment size to be reduced.

The SIMULIA Co-Simulation Services provides multiple negotiation methods as described below.

Constant Coupling Step Size

A constant user-defined coupling step size is used to determine the next coupling step size. In this case, the solver's preferred solver coupling step sizes are ignored. All analyses advance while exchanging data at target points according to

$$t_{i+1} = t_i + \Delta t_c,$$

where Δt_c is a value that defines the coupling step size to be used throughout the coupled simulation, t_{i+1} is the target time, and t_i is the time at the start of the coupling step. This strategy is simple; however, it is not ideal when the individual solvers adapt their increment sizes for stability and accuracy.

Minimum Coupling Step Size

This method selects the minimum of the preferred step sizes suggested by each analysis. The minimum method cannot be used in a co-simulation involving Abaqus/Explicit.

Maximum Coupling Step Size

This method selects the maximum of the preferred step sizes suggested by each analysis.

Having a Solver Dictate the Coupling Step Size

In this method a designated solver specifies the coupling size; Abaqus ignores the other solver's preferred coupling step sizes. You can use this method only with Abaqus/Explicit and only when Abaqus/Explicit dictates the coupling step size.

Constant Multiple

This method is similar to the constant coupling step size method, except that the coupling step size is dictated by one of the solvers at the start of the analysis. In addition, the other clients' preferred step size is used to determine the constant multiple of the dictated coupling step size.

The rendezvousing scheme and its associated parameters are defined in the SIMULIA Co-Simulation Services configuration file. For guidance in selecting an appropriate rendezvousing scheme for co-simulation between Abaqus solvers, see *Co-Simulation between Abaqus Solvers*. For co-simulation between Abaqus and third-party analysis programs, see the appropriate user’s guide.

Using the SIMULIA Co-Simulation Services Configuration File

The SIMULIA Co-Simulation Services employs an independent software component, termed the "Director," which defines all aspects of the coupled simulation. You provide the Director with relevant parameters through the CSS Configuration file.

The CSS Configuration file must be in Extensible Markup Language (XML) format, which uses the file extension `xml`, and is detailed in *About the SIMULIA Co-Simulation Services Configuration File*.

In most cases, you will not need to author CSSConfiguration file; most commonly it is generated by a pre-processing tool or through a predefined template.

For the co-simulation analysis cases described in *Co-Simulation between Abaqus Solvers*, there are predefined templates, as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
Required XML declaration line
<CoupledMultiphysicsSimulation>
Required XML root element; identifies file as describing a multiphysics simulation
    <template_name>
        <template_parameter_1>parameter_1_name</template_parameter_1>
        <template_parameter_2>parameter_2_name</template_parameter_2>
        <template_parameter_3>parameter_3_name</template_parameter_3>
    </template_name>
    Closure of the template element
</CoupledMultiphysicsSimulation>
Closure of the XML root element
```

You enter the template name and a short list of parameter settings, such as the names of the two analysis jobs and the duration of the analysis. Details of the predefined templates are provided in *Structural-to-Structural Co-Simulation* and *Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation*, as well as information on how to obtain an example configuration file for each template. At run time, the SIMULIA Co-Simulation Services director applies your parameter settings to the template, creating an elaborated Configuration file.

In cases where predefined templates are not available (such as coupling with an in-house or third-party code) or are insufficient (for example, you want to exchange more variables at the co-simulation interface region or adjust mapping tolerances), you must either modify an existing Configuration file or create a new Configuration file.

Model Dimension and Coordinate Systems

Two-dimensional and three-dimensional Abaqus models are fully supported. Axisymmetric Abaqus models are supported only for Abaqus/Standard to Abaqus/Explicit co-simulation. For co-simulations that do not support two-dimensional and axisymmetric models, you can represent these models as a three-dimensional slice of unit thickness (or wedge element) with the appropriate boundary conditions applied.

Vector quantities are defined according to Abaqus conventions; the first component represents the quantity along the x-axis, the second quantity represents the quantity along the y-axis, and the third quantity represents the quantity along the z-axis (for three-dimensional models). For axisymmetric models in Abaqus the axis of revolution is about the y-axis. These conventions apply to both the exported and the imported vector quantities.

All exported vector quantities are expressed in the global coordinate system of the Abaqus model, ignoring any transformation definitions. Similarly, the third-party program must provide vector quantities that are imported into Abaqus in the global coordinate system of the Abaqus model.

The third-party analysis program might use different conventions, please refer to the appropriate third-party program documentation for further modeling details and/or limitations.

Unit System

Abaqus does not require that the analysis be run with a particular unit system. In general, the unit system used in creating the Abaqus model might not be the same as that used with the third-party program model. When the units systems differ, the fields exchanged between the two programs must go through a transformation of units. There are two possible scenarios of who performs the unit translation. Refer to the appropriate third-party program documentation for further modeling details on units system translation.

Units Translation Performed by SIMULIA Co-Simulation Services

The units system translation can be done by the SIMULIA Co-Simulation Services. In this case all solvers must support units translation through the SIMULIA Co-Simulation Services, and you must declare the units system in the Abaqus model (*Units*). Units translation can be performed only for field types that have a predefined dimensionality (for example, the velocity field has a predefined dimensionality of LT^{-1}). Field variables in Abaqus do not have a predefined dimensionality; therefore, unit translation is not performed for field variables.

Units Translation Performed by client

In some cases unit translation is done by the third-party solver. In this case you do not need to declare the units system in the Abaqus model.

Restarting a Co-Simulation

Restart output must be synchronized between co-simulation analyses for a co-simulation restart to be successful. You should request that restart data are written at the co-simulation target times when both solutions are considered at equilibrium. For more information, see *Synchronizing Restart Information Written in a Co-Simulation*. The solution time for the particular step/increment number from which Abaqus restarts must correspond to the coupled analysis solution.

Limitations

The following limitations apply:

- The steps in the Abaqus model must be defined such that the co-simulation fits entirely within a single Abaqus step. Further, there can be only one co-simulation in the Abaqus job. You can use the restart capability to perform multiple co-simulations for an analysis (see *Restarting an Analysis*).
- A co-simulation surface or volume defined over beam, pipe, and truss elements or defined over the edges of three-dimensional elements cannot be used as an interface region. You should use discrete points to transfer loads and boundary conditions.
- A co-simulation surface or volume defined over modified triangular elements or modified tetrahedral elements cannot be used as an interface region.
- Quadratic coupled temperature-displacement elements cannot be used as an interface region in a co-simulation using the coupled temperature-displacement procedure.
- When performing a co-simulation, output at specified time points might not be satisfied at the requested times, depending on the synchronization parameters.
- Abaqus/Explicit cannot use thread-based execution with co-simulation.

Because co-simulation is a general capability, no error message is issued if Abaqus encounters the conditions listed above. There might be further limitations depending on the third-party analysis program being used. For more information, refer to the appropriate third-party program documentation.

Co-Simulation between Abaqus Solvers

In this section:

- *Structural-to-Structural Co-Simulation*
- *Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation*
- *Executing a Co-Simulation*

Structural-to-Structural Co-Simulation

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Co-Simulation*
- *Preparing an Abaqus Analysis for Co-Simulation*
- **CO-SIMULATION*

Overview

Co-simulation between two structural solvers (solvers exchanging displacements and rotations and the conjugate fields forces and moments) represents a very strong physics coupling and requires special treatment at the co-simulation interface. Abaqus supports Abaqus/Standard to Abaqus/Explicit co-simulation and Abaqus to Simpack (a multibody dynamics solver) co-simulation by providing specialized interface handling.
Although you can perform a mechanical-to-mechanical co-simulation between two Abaqus/Standard analyses or between two Abaqus/Explicit analyses, it is not recommended due to the lack of proper numerical handling at the interface.

Abaqus supports three interface handling methods: the enhanced subcycling method, the "original" subcycling method, and the lockstep method. The enhanced subcycling method is the preferred method, providing robust and accurate solutions in the most cost effective manner. This method also allows constraints to be located on the co-simulation interface. The latter two methods are documented for legacy purposes.

This section discusses analysis setup, execution, and limitation details specific to Abaqus/Standard to Abaqus/Explicit co-simulation.

Refer to *Dynamic impact of a scooter with a bump* for an example of Abaqus/Standard to Abaqus/Explicit co-simulation.

Coupling in Abaqus/Standard and Abaqus/Explicit Co-Simulation

Co-simulation between Abaqus/Standard and Abaqus/Explicit is a multiple domain analysis approach, where each Abaqus analysis operates on a complementary section of the model domain where it is expected to provide the more computationally efficient solution. For example, Abaqus/Standard provides a more efficient solution for light and stiff components, while Abaqus/Explicit is more efficient for solving complex contact interactions. In addition, when a portion of the Abaqus/Standard model is linear, you can use substructures to further reduce the computational cost.

You can use the following Abaqus/Standard analysis procedures:

- *Static Stress Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*

You can use the following Abaqus/Explicit analysis procedure:

- *Explicit Dynamic Analysis*

Identifying the Co-Simulation Interface Region

The domains modeled in each Abaqus analysis are complementary, and the interaction between the models takes place through a common interface region. You can specify an interface region using either node sets or surfaces when coupling Abaqus/Standard to Abaqus/Explicit. However, you must be consistent in your region definition

in Abaqus/Standard and Abaqus/Explicit. If you define a co-simulation region with a node set or node-based surface in one analysis, you must use the same type of co-simulation region definition in the other analysis. For node-based surfaces the nodes must be coincident because no topology information is provided to conservatively map fields between the models. Likewise, if you define a co-simulation region with an element-based surface in one analysis, you must define your co-simulation region with an element-based surface in the other analysis.

You can have dissimilar meshes in regions shared in the Abaqus/Standard and Abaqus/Explicit model definitions when using element-based surfaces. In some cases you can improve solution stability and accuracy by ensuring that you have matching nodes at the interface (see *Dissimilar Mesh-Related Limitations*). In these cases you can use the modeling practice described in *Ensuring matching nodes at the interface regions* to ensure these matching nodes.

In the step definition, you declare a co-simulation step, define the interface region, and specify the solution fields to exchange. When coupling Abaqus/Standard to Abaqus/Explicit, you can declare only a single interface region; if there are multiple regions, you must combine them into a single region. The fields exchanged depend on the interface method and whether or not rotational degrees of freedom are active at the interface.

Input File Usage: Use the following options to define an element-based or node-based surface as a co-simulation region:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=SURFACE (default)
element-based or node-based surface, *comma-separated list of fields*
**CO-SIMULATION REGION*, EXPORT, TYPE=SURFACE (default)
element-based or node-based surface, *comma-separated list of fields*

Use the following options to define a node set as a co-simulation region:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE
node set, *comma-separated list of fields*
**CO-SIMULATION REGION*, EXPORT, TYPE=NODE
node set, *comma-separated list of fields*

Enhanced Subcycling Method for Interface Calculations

The enhanced subcycling method is the preferred method for coupling Abaqus/Standard to Abaqus/Explicit. This method ensures velocity compatibility at the interface, where calculations are performed by an interface service; stiffness and mass properties of the underlying structure are exchanged in the form of dynamic interface operators that provide robust and accurate solutions for strongly coupled physics. Abaqus/Explicit subcycles (that is, takes one or more increments) per Abaqus/Standard increment to ensure a cost-effective solution for both domains. This method allows constraints to exist on the co-simulation interface region. *Table 1* and *Table 2* describe the fields exchanged and their causality (whether the field is imported or exported). Fields associated with a rotational degree of freedom should be exchanged only when the elements underlying the interface are structural elements with rotational degrees of freedom.

Table 1: Fields imported and exported by Abaqus/Standard for the enhanced subcycling method.

Field	Description	Causality
IFORCE	Interface force used by the interface service.	Import
IMOMENT	Interface moment used by the interface service when rotational degrees of freedom are active.	Import
VT	Translational velocity.	Export

Field	Description	Causality
VR	Rotational velocity when rotational degrees of freedom are active.	Export
H	Dynamic interface operator used by the interface service; the dynamic operator is constructed based on whether or not rotational degrees of freedom are active.	Export

Table 2: Fields imported and exported by Abaqus/Explicit for the enhanced subcycling method.

Field	Description	Causality
VTINTRF	Translation velocity used by the interface service.	Import
VRINTRF	Rotational velocity used by the interface service when rotational degrees of freedom are active.	Import
H	Dynamic interface operator used by the interface service; the dynamic operator is constructed based on whether or not rotational degrees of freedom are active.	Import
IFORCE	Interface force used by the interface service.	Export
IMOMENT	Interface moment used by the interface service when rotational degrees of freedom are active.	Export

An interface solve is performed in Abaqus/Explicit for every Abaqus/Explicit increment. This solve can be costly because the interface matrix used for the interface solve is dense, and its size scales with the number of interface nodes.

Input File Usage:

Use the following options in the Abaqus/Standard model with active rotational degrees of freedom at the interface for the enhanced subcycling method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE
node set or surface name, IFORCE, IMOMENT
**CO-SIMULATION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, VT, VR, H

Use the following options in the Abaqus/Explicit model with active rotational degrees of freedom at the interface for the enhanced subcycling method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE
node set or surface name, VTINTRF, VRINTRF
**CO-SIMULATION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, IFORCE, IMOMENT

Original Subcycling Method for Interface Calculations

Similar to the enhanced subcycling method, the "original" subcycling method ensures velocity compatibility at the interface and provides a robust and accurate solution for strong coupled physics.

Table 3 and *Table 4* describe the fields exchanged and their causality. You must specify rotational fields even if they are not active at the interface.

Table 3: Fields imported and exported by Abaqus/Standard for the original subcycle method.

Field	Description	Causality
VRINTRF	Rotational velocity used by the interface service.	Import
MASSINV	Inverse mass operator.	Import
RICUR	Rotary inertia.	Import
IFORCE	Interface force computed by the interface service.	Export
IMOMENT	Interface moment computed by the interface service.	Export
VT	Translation velocity used for initial conditions.	Export
VR	Rotational velocity used for initial conditions.	Export
VTINTRF	Translational velocity used by the interface service.	Import

Table 4: Fields imported and exported by Abaqus/Explicit for the original interface method.

Field	Description	Causality
IFORCE	Interface force computed by the interface service.	Import
IMOMENT	Interface moment computed by the interface service.	Import
VTINIT	Initial translational velocity.	Import
VRINIT	Initial rotational velocity.	Import
VT	Translational velocity.	Export
VR	Rotational velocity.	Export
MASS	Lumped mass.	Export
RICUR	Rotary inertia.	Export

An interface solve is performed in Abaqus/Standard for every Abaqus/Explicit increment. This solve can be costly for two reasons. First, the interface matrix used for the interface solve is dense, and its size scales with the number of interface nodes. Second, the interface matrix changes every Abaqus/Explicit increment, requiring factorization in Abaqus/Standard for every Abaqus/Explicit increment.

Input File Usage: Use the following options in the Abaqus/Standard model for the original subcycling method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE
node set or surface name, VTINTRF, VRINTRF, MASSINV, RICUR
**CO-SIMULATION REGION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, IFORCE, IMOMENT, VT, VR

Use the following options in the Abaqus/Explicit model for the original subcycling method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE

node set or surface name, IFORCE, IMOMENT, VTINIT, VRINIT
**CO-SIMULATION REGION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, VT, VR, MASS, RICUR

Lockstep Method for Interface Calculations

The time incrementation that you choose for coupling affects the solution computational cost, accuracy, and stability. The enhanced subcycling method is frequently the more robust, accurate, and cost effective method because Abaqus/Standard time increments, free of any forced co-simulation time incrementation constraints, are commonly much longer than Abaqus/Explicit time increments. However, this subcycling method might be less cost effective when a large portion of the nodes in the model are at the co-simulation interface. This is because stabilization operations at the interface (a "free solve") for each increment in the Abaqus/Explicit analysis is performed. These free-solve operations require an implicit solution of a dense system of equations that scale with the number of interface nodes. In cases with a large number of interface nodes, the computational cost of this interface solve can exceed any cost savings seen due to subcycling. Therefore, for a model where a significant share of the nodes are at the co-simulation interface, performance might be poorer with the subcycling scheme. In such a case, you can use the lockstep method, where Abaqus/Standard is forced to use the time increment size of the Abaqus/Explicit analysis. This approach enforces displacement compatibility at the interface.

Table 5 and *Table 6* describe the fields exchanged and their causality. You must specify rotational fields even if there are no rotational degrees of freedom active at the interface.

Table 5: Fields imported and exported by Abaqus/Standard for the lockstep method.

Field	Description	Causality
CF	Concentrated force.	Import
CM	Concentrated moment.	Import
MASS	Lumped mass.	Import
RICUR	Rotary inertia.	Import
UTPRED	Predictor displacement.	Export
URPRED	Predictor rotation.	Export
VT	Translational velocity.	Export
VR	Rotational velocity.	Export
AT	Translational acceleration.	Export
AR	Rotational acceleration.	Export

Table 6: Fields imported and exported by Abaqus/Explicit for the lockstep method.

Field	Description	Causality
UTINIT	Initial displacements.	Import
URINIT	Initial rotations.	Import
VTINIT	Initial translational velocity.	Import
VRINIT	Initial rotational velocity.	Import
ATINIT	Initial translational acceleration.	Import
ARINIT	Initial rotational acceleration.	Import
UT	Displacement.	Import
UR	Rotation.	Import

Field	Description	Causality
CF	Concentrated forces.	Export
CM	Concentrated moments.	Export
MASS	Mass.	Export
RICUR	Rotary inertia.	Export

Input File Usage: Use the following options in the Abaqus/Standard model for the lockstep method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE
node set or surface name, CF, CM, MASS, RICUR
**CO-SIMULATION REGION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, UTPRED, URPRED, VT, VR, AT, AR

Use the following options in the Abaqus/Explicit model for the lockstep method:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=NODE *or* SURFACE
node set or surface name, UTINIT, URINIT, VTINIT, VRINIT, ATINIT, ARINIT
node set or surface name, UT, UR
**CO-SIMULATION REGION*, EXPORT, TYPE=NODE *or* SURFACE
node set or surface name, CF, CM, MASS, RICUR

Creating a Configuration File

You can use predefined templates to create a configuration file for the coupling schemes described above. *Table 7* describes the predefined templates available for Abaqus/Standard to Abaqus/Explicit co-simulation and lists example configuration files that you can review.

Table 7: Templates for structural-to-structural co-simulation.

Enhanced subcycling method	Coupling scheme: Allow Abaqus/Explicit to subcycle. This is the recommended method.
	`template_std_xpl_subcycleEnhanced`
	Example file: `exa_std_xpl_subcycleEnhanced.xml`
Original subcycling method	Coupling scheme: Allow Abaqus/Explicit to subcycle.
	`template_std_xpl_subcycle`
	Example file: `exa_std_xpl_subcycle.xml`
Lockstep method	Coupling scheme: Abaqus/Standard and Abaqus/Explicit use a single increment per coupling step (lockstep).
	`template_std_xpl_lockstep`
	Example file: `exa_std_xpl_lockstep.xml`

To obtain an example configuration file, you can use the **abaqus fetch** utility. For example, to obtain the example for Abaqus/Standard to Abaqus/Explicit subcycling, use the following command:

```
abaqus fetch job=exa_std_xpl_subcycleEnhanced.xml
```

You can then modify the configuration by modifying the replaceable text (rotation degrees of freedom active at interface, job names, Abaqus/Explicit interface region name and duration).

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
```

```
   <template_std_xpl_subcycleEnhanced rotation="true or false">
      <Standard_Job>standard_job_name</Standard_Job>
      <Explicit_Job>explicit_job_name</Explicit_Job>
      <Explicit_Interface>region_name</Explicit_Interface>
      <duration>duration_value</duration>
   </template_std_xpl_subcycleEnhanced>
</CoupledMultiphysicsSimulation>
```

In certain cases you might need to use co-simulation configuration features that are not described in the predefined templates. For example, you might want to change the dissimilar mesh mapping search tolerances; these tolerances are available generally in the configuration file but are not described in the predefined templates. For these cases, you must create an elaborated configuration file.

Executing the Coupled Analysis

You execute the co-simulation interactively in Abaqus/CAE or from the command line, as described in *Executing a Co-Simulation*.

Diagnostics Information

The Abaqus/Standard job provides detailed descriptions of co-simulation operations in the message (`.msg`) file. For the subcycling scheme the status (`.sta`) file provides summary information indicating when the interface calculations followed by a re-solve of the increment are made, as shown in the following example status file. The `E` suffix in the attempt-count entry (column 3) indicates an increment performing interface calculations. An increment without the `E` suffix indicates a re-solve of the increment.

```
SUMMARY OF JOB INFORMATION:
STEP   INC ATT SEVERE EQUIL TOTAL  TOTAL      STEP       INC OF       DOF
  IF
               DISCON ITERS ITERS  TIME/    TIME/LPF    TIME/LPF    MONITOR
RIKS
                ITERS               FREQ
  1     1   1E    0     1     1  0.000       0.000      0.001000
  1     1   1     0     3     3  0.00100     0.00100    0.001000
  1     2   1E    0     1     1  0.00100     0.00100    0.001000
  1     2   1     0     3     3  0.00200     0.00200    0.001000
  1     3   1E    0     1     1  0.00200     0.00200    0.001000
  1     3   1     0     2     2  0.00300     0.00300    0.001000
  1     4   1E    0     1     1  0.00300     0.00300    0.001000
  1     4   1     0     3     3  0.00400     0.00400    0.001000
```

The Abaqus/Explicit job provides summary descriptions of co-simulation operations in the status (`.sta`) file.

Limitations

The following limitations apply to Abaqus/Standard to Abaqus/Explicit co-simulation in addition to the limitations discussed in *Preparing an Abaqus Analysis for Co-Simulation*.

General Limitations

- Displacement compatibility at the co-simulation interface is not maintained when you choose either subcycling solution method. For these methods, velocity compatibility is maintained, but you might see small amounts of displacement mismatch between Abaqus/Standard and Abaqus/Explicit as the simulation advances in time. This “drift” is more pronounced if severe nonlinearity (such as plastic deformation) occurs at the co-simulation interface. You can control this drift by adjusting Abaqus/Standard solution parameters so that the Abaqus/Standard increment size is reduced (for example, by limiting the maximum time increment size or specifying a smaller half-increment residual tolerance for implicit dynamic analyses).

- Nodal transformations are not permitted on the co-simulation region nodes.
- The ALE technique might not be used in elements attached to co-simulation region nodes.
- Fully coupled temperature-displacement elements can be used, but no temperature quantities are exchanged.
- An Abaqus/Standard static stress analysis cannot be used with the lockstep time incrementation scheme in Abaqus/Standard to Abaqus/Explicit co-simulation.
- Only points and surface regions are allowed; coupling volume regions is not supported.
- Only a single region can be defined as the interface region; multiple interface regions are not supported.
- Units system conversion is not supported with the enhanced subcycling algorithm because the dynamic operator contains mixed dimensionality terms for translation and rotations.

Dissimilar Mesh-Related Limitations

When your Abaqus/Standard and Abaqus/Explicit co-simulation region meshes differ, the following limitations apply:

- Solution accuracy might be affected when your co-simulation region meshes are not uniform in the presence or absence of rotational degrees of freedom; for example, if a continuum element mesh is locally reinforced with beam or shell elements at the co-simulation region interface.
- In cases where the stress state near the co-simulation interface is significant (approaching 1% or more) relative to the material stiffness, you might observe appreciable irregular mesh distortion if the mesh density adjacent to the co-simulation region differs greatly between the Abaqus/Explicit and Abaqus/Standard models. For example, this effect is common with large deformation of hyperelastic materials. You can minimize this effect by choosing a similar or finer mesh at the Abaqus/Standard co-simulation region when using the subcycling time integration scheme or by choosing a similar or finer mesh at the Abaqus/Explicit co-simulation region when using the lockstep time integration scheme.

Abaqus/Standard Analysis Limitations

Abaqus/Standard elements that have no equivalent degree-of-freedom counterpart in Abaqus/Explicit cannot be connected to co-simulation region nodes. These elements include

- Axisymmetric elements with twist degrees of freedom (the CGAX element family)
- Axisymmetric solid elements with asymmetric deformation (the CAXA element family)
- Generalized plane strain elements (the CPEG element family)
- Coupled pore pressure-displacement elements
- Heat transfer and thermal-electrical elements
- Acoustic elements
- Piezoelectric elements

The following specific limitations also apply:

- A co-simulation region node cannot be a secondary node in a tie constraint, an MPC constraint, or a kinematic coupling constraint.

Abaqus/Explicit Analysis Limitations

Stability and accuracy of the co-simulation solution might be adversely affected when the following model features are defined at or near the co-simulation region:

- Connector elements connected to co-simulation region nodes.
- Co-simulation region nodes that participate in a tie constraint, an MPC constraint, or a kinematic coupling constraint.

When using these features, you should compare the Abaqus/Standard and Abaqus/Explicit solutions (for example, compatibility of the displacement history) at the co-simulation interface as an indicator of solution accuracy.

Electromagnetic-to-Structural and Electromagnetic-to-Thermal Co-Simulation

Products: Abaqus/Standard

References:

- *About Co-Simulation*
- *Preparing an Abaqus Analysis for Co-Simulation*
- **CO-SIMULATION*

Overview

This section discusses analysis setup and execution details specific to electromagnetic-to-structural and electromagnetic-to-thermal co-simulation using Abaqus/Standard procedures. Co-simulation between a time-harmonic or transient electromagnetic analysis and a static, transient implicit dynamic, coupled temperature-displacement, or transient heat transfer analysis is supported. You can also solve electromagnetic-to-structural problems by coupling Abaqus/Standard with CST Studio Suite or coupling Abaqus/Standard to a third-party electromagnetic solver (for more information, see the respective user's guide).

An electromagnetic to a heat transfer co-simulation analysis is useful for applications such as induction heating, which involves two-way coupling: the Joule heat production from the electromagnetic analysis drives a heat transfer analysis and determines the temperature distribution, while the temperature distribution, in turn, affects the electromagnetic fields through temperature-dependent material properties (such as electrical conductivity and magnetic permeability). For electromagnetic-to-thermal coupling, co-simulation between a time-harmonic or transient electromagnetic analysis and a transient heat transfer analysis is supported.

An electromagnetic to transient implicit dynamic analysis is useful for applications such as electromagnetic forming, where the Lorentz body forces from an electromagnetic analysis drive a transient dynamic analysis. Co-simulation between a transient electromagnetic analysis and a static or transient implicit dynamic analysis is supported. However, the coupling is only one way; that is, the effects of deformation of parts of the domain (metal work piece, in this case) on the electromagnetic fields is not accounted for. Hence, such analysis should be used only when the effects of deformation on the electromagnetic fields are relatively small.

Identifying the Abaqus Step for the Co-Simulation Analysis

The following Abaqus/Standard analysis procedures can be used for an electromagnetic-to-structural co-simulation:

- *Static Stress Analysis*
- *Implicit Dynamic Analysis Using Direct Integration*
- *Eddy Current Analysis*

The following Abaqus/Standard analysis procedures can be used for electromagnetic-to-thermal co-simulation:

- *Uncoupled Heat Transfer Analysis*
- *Eddy Current Analysis*

Input File Usage: Use the following option within a step definition for an Abaqus/Standard to Abaqus/Standard co-simulation:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS

Identifying the Co-Simulation Interface Region

Interaction between the electromagnetic and structural models occurs through a common volume interface region.

You must specify the volume interface region using element sets between the Abaqus/Standard analyses. You must be consistent in your region definition in both the Abaqus/Standard simulations. In other words, the regions must be co-located and have the same extent; however, they can have different element topology.

Input File Usage: Use the following option to define an element-based co-simulation region in an Abaqus/Standard model:

**CO-SIMULATION REGION*, TYPE=VOLUME
elset_A

Only one **CO-SIMULATION REGION* option can be defined in each Abaqus/Standard analysis. In addition, only one element set can be defined.

Identifying the Fields Exchanged across a Co-Simulation Interface

For Abaqus/Standard to Abaqus/Standard co-simulation, see the tables in *Identifying the Fields Exchanged across a Co-Simulation Interface* for lists of fields that are available for co-simulation exchange.

In an electromagnetic to heat transfer co-simulation, the fields exchanged are temperature (NT), Joule heating flux due to current flow (EMJH), and concentrated heat flux (CFLUX).

Input File Usage: Use the following options in the electromagnetic model to perform an electromagnetic-to-heat transfer co-simulation:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=VOLUME
elset name, NT
**CO-SIMULATION*, EXPORT, TYPE=VOLUME
elset name, EMJH

Use the following options in the heat transfer model to perform an electromagnetic-to-heat transfer co-simulation:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=VOLUME
elset name, CFLUX
**CO-SIMULATION*, EXPORT, TYPE=VOLUME
elset name, NT

In an electromagnetic-to-structural co-simulation, a one-way co-simulation is performed with the electromagnetic model exporting magnetic body force (EMBJ) and the structural model importing concentrate forces (CF) at nodes.

Input File Usage: Use the following options in the electromagnetic model to perform an electromagnetic-to-structural co-simulation:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, EXPORT, TYPE=VOLUME
elset name, EMBJ

Use the following options in the structural model to perform an electromagnetic-to-structural co-simulation:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS
**CO-SIMULATION REGION*, IMPORT, TYPE=VOLUME
elset name, CF

Defining the Coupling and Rendezvousing Scheme

The SIMULIA Co-Simulation Services configuration file is used to define the time incrementation process and the frequency of exchange between the two Abaqus/Standard analyses.

Predefined templates are available for commonly used coupling schemes. You can refer to these templates when you create your configuration files. This section describes the rendezvous scheme settings and the predefined configuration file templates.

Coupling Scheme

The sequential explicit coupling scheme (also referred to as the Gauss-Seidel coupling algorithm) and the iterative coupling scheme are available for electromagnetic-to-structural and electromagnetic-to-thermal co-simulation. The electromagnetic analysis must always lead the co-simulation, while the heat transfer or the stress analysis always lags the co-simulation. All the predefined templates are set up with the above lead-lag sequence.

Time Incrementation Scheme

You can force the two transient Abaqus/Standard analyses to use the same increment size, or you can allow the increment sizes to differ (subcycling). The time incrementation scheme that you choose for coupling affects the solution computational cost and accuracy. When using the subcycling method, this data exchange does not represent a constraint on Abaqus/Standard incrementation; the Abaqus/Standard analysis advances in time using its normal time incrementation logic but performs data exchanges as needed at the coupling step size intervals.

A time-harmonic electromagnetic procedure is defined in the frequency domain and does not have a solution time scale associated with it in the sense that a transient analysis does. It is convenient to introduce a pseudo-solution time scale that is associated with the time-harmonic electromagnetic procedure involved in a co-simulation analysis, thereby facilitating coupling with a transient analysis at certain solution time intervals of the latter analysis. The pseudo-time scale of the time-harmonic electromagnetic analysis follows the solution time scale in the transient heat transfer analysis and is reset in every coupling step in a manner described below.

Coupling Step Size

The coupling step size is the period between two consecutive co-simulation data exchanges between the two Abaqus/Standard analyses. For transient electromagnetic to transient heat transfer or transient implicit dynamic co-simulation, the coupling step size can be specified to be equal to the minimum of the time step sizes determined by the automatic time incrementation schemes of the individual analyses or to a constant user-defined value.

When the leading electromagnetic analysis is time harmonic, the coupling step size can be specified to be equal to the time step size of the lagging transient heat transfer or implicit dynamic analysis or to a constant user-defined value. In the latter case, the time-harmonic electromagnetic analysis would solve for the fields at the end of each successive constant user-defined coupling step size, while the lagging heat transfer or stress analysis would typically subcycle until the target coupling step time is reached.

For iterative coupling, the two analyses must be coupled at the end of each time increment, and subcycling should not be used. If subcycling is used in this situation, the exchanged updated solutions during the iterations will be utilized only for the very last increment and the cumulative effect of the updates over the previous increments (between the last coupling and the current coupling) will be lost.

Creating a Configuration File

You can use predefined templates to create the configuration file for the coupling schemes described above. *Table 1* describes the predefined templates available for electromagnetic to transient heat transfer analyses and for electromagnetic to stress-displacement analyses and lists example configuration files that you can review.

Table 1: Templates for electromagnetic co-simulation.

Electromagnetic to transient heat transfer co-simulation	Coupling scheme: • Electromagnetic analysis leads • Heat transfer analysis defines the coupling step size
	`template_em_std_export`
	Example file: `exa_em_std_export`
	Coupling scheme: • Electromagnetic analysis leads • Allow heat transfer analysis to subcycle
	`template_em_std_fixed`
	Example file: `exa_em_std_fixed`
	Coupling scheme: • Electromagnetic analysis leads • Heat transfer analysis defines the coupling step size • Iterative coupling
	`template_em_std_iterative`
	Example file: `exa_em_std_iterative`
Electromagnetic to Abaqus/Standard stress/displacement co-simulation	Coupling scheme: • Electromagnetic analysis leads • Step size is determined based on the minimum suggested step size of the electromagnetic and Abaqus/Standard stress/displacement analyses. • Either analysis can subcycle • Body forces are transferred from the electromagnetic to Abaqus/Standard stress/displacement analysis. No other co-simulation transfer occurs.
	`template_em_std_force_oneway`
	Example file: `exa_em_std_force_oneway`

To obtain an example configuration file, you can use the **abaqus fetch** utility. For example, to obtain the example for which the heat transfer analysis serves as the main analysis in determining the coupling step size, use the following command:

```
abaqus fetch job=exa_em_std_export
```

The example file `exa_em_std_export.xml` is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
   <template_em_std_export>
      <EM_Job>em_job_name</EM_Job>
```

```
      <HeatTransfer_Job>ht_job_name</HeatTransfer_Job>
      <duration>duration_value</duration>
   </template_em_std_export>
</CoupledMultiphysicsSimulation>
```

In certain cases you may need to use co-simulation configuration features that are not described in the predefined templates. For example, you may wish to change the dissimilar mesh mapping search tolerances; these tolerances are available generally in the configuration file but are not described in the predefined templates. For these cases, you must create an elaborated configuration file.

Executing the Coupled Analysis

You execute the co-simulation from the command line, as described in *Executing a Co-Simulation*.

Limitations

The limitations discussed in *Preparing an Abaqus Analysis for Co-Simulation* apply to electromagnetic-to-structural and electromagnetic-to-thermal co-simulation.

Executing a Co-Simulation

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*
- *Understanding co-executions*

Overview

You can execute the following types of co-simulations from the command line:

- Structural-to-structural
- Electromagnetic-to-structural
- Electromagnetic-to-thermal
- Structural-to-logical

You can also execute structural-to-structural co-simulations in Abaqus/CAE.

Executing a Co-Simulation from Abaqus/CAE

You can execute the coupled analysis interactively in Abaqus/CAE as described in *Understanding co-executions*. You are not required to create a configuration file; Abaqus/CAE creates the file automatically.

Abaqus/CAE Usage: Job module:
Co-execution->**Create**: select the Abaqus/Standard model and the Abaqus/Explicit model; **Communication time out**: *timeout-value*
Co-execution->**Manager**: **Submit**

Executing a Co-Simulation from the Command Line

You execute the Abaqus jobs as described in *Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*.

Command Usage Example

Use the following command to submit a co-simulation between two Abaqus analyses, “job-1” and “job-2”:

```
abaqus cosimulation cosimjob=beam job=job-1,job-2
configure=config
```

Considerations for Using the timeout Parameter

The **timeout** execution parameter specifies the amount of time in seconds that each analysis waits to receive the co-simulation message expected from the other analysis that is running. The default timeout value is 60 minutes when submitting jobs using the command line options and 10 minutes when executing the jobs in Abaqus/CAE. When the timeout period is large compared to typical analysis increment wallclock times, you have greater flexibility in starting jobs and performing operations that precede the co-simulation analysis step. Examples where this flexibility is needed include: job submission using queues, analyses where steps that precede the co-simulation step have long run times, and cases where one job is resubmitted because of an input error. However,

a large timeout period can cause problems when one of the co-simulation jobs fails (for reasons such as convergence issues or availability of computer resources) before the initial co-simulation communication is established. In these cases you may prefer to terminate the job left running rather than have it wait the entire timeout period.

Limitations

Electromagnetic-to-structural and electromagnetic-to-thermal co-simulations are not supported in Abaqus/CAE.

System-Level Modeling between Logical and Physical Interactions

Products: Abaqus/Standard Abaqus/Explicit

References:

- *SIMULIA Co-Simulation Services FMU Execution*
- *About Co-Simulation*
- *Preparing an Abaqus Analysis for Co-Simulation*
- **CO-SIMULATION*
- **AMPLITUDE*
- **OUTPUT*

Overview

This section discusses applications, analysis setup, and execution details for system-level modeling between logical-to-physical co-simulation using the SIMULIA Co-Simulation Services. The logical model is a Functional Mockup Unit (FMU) conforming to the Functional Mockup Interface (FMI) defined by the Modelisar organization.

For more information on the FMI standard or for a list of tools that provide the FMU file export capability, refer to `http://fmi-standard.org/`. Although you can use FMUs with nonstructural applications, this section focuses on coupling logical models with the Abaqus structural solvers.

Applications

System-level modeling refers to the modeling of systems that can include both physical (structural, thermal, acoustics, computational fluid dynamics, etc.) and logical components. Logical modeling refers to a large class of modeling abstractions often encountered in the engineering practice. Generally speaking, you can designate a part of a system as using a logical modeling abstraction when most (if not all) of the geometry of the part is removed. Physical modeling is the complementary modeling abstraction to logical modeling. Abaqus uses a physical modeling abstraction most of the time; as elements deform, they know precisely about their geometry, thus trying to mimic the real world at a fine-grain level. In many engineering systems the interaction between logical and physical components is paramount, and you cannot fully analyze one without the other.

System-level modeling can be used to couple:

- Dymola with Abaqus,
- FMUs conforming to the FMI 2.0 standard with Abaqus,
- multiple FMUs with one another, and
- FMUs with models created using the Physics Simulation apps on the **3D**EXPERIENCE platform (including structural and computational fluid dynamic models). The models must support the SIMULIA Co-Simulation Services and signal support.

This functionality is not intended for applications where a strong interaction between subsystems exists. For example, results obtained for a coupled analysis in which the tires and the road are modeled in Abaqus while the suspension and the body are modeled as an FMU will likely lead to unstable results.

Preparing an Abaqus Model for a Logical-to-Physical Co-Simulation

To prepare an Abaqus model for a logical-to-physical co-simulation, you need to:

- identify the Abaqus analysis step,

- define sensors, and
- define actuators.

The following sections describe these steps. For more detailed information on co-simulation analysis model creation, see *Preparing an Abaqus Analysis for Co-Simulation*.

Identifying the Abaqus Analysis Step

The period of time of co-simulation interaction is defined as a co-simulation event. The co-simulation event begins at the start of an Abaqus analysis step and ends by the end of that step; the analysis step is referred to as the co-simulation step. See *Identifying an Abaqus Step for Co-Simulation Analysis* for the analysis procedures that support co-simulation. An analysis job can contain only one co-simulation step. You can use the restart capability to define additional co-simulation steps.

You can define a co-simulation step where Abaqus exchanges signals with one or more FMUs.

Input File Usage: Use the following option within a step definition:

**CO-SIMULATION*, PROGRAM=MULTIPHYSICS

Defining Sensors

A sensor is a scalar quantity that can be exported by Abaqus. A sensor can be any nodal or connector element history output variable defined as a named sensor (see *Defining Sensors*). In addition, some of the whole surface contact and contact pair output history variables can be defined as a named sensor.

Defining Actuators

An actuator is a scalar quantity imported into Abaqus. It acts as an amplitude and can actuate any Abaqus feature that can reference an amplitude, such as concentrated loads, boundary conditions, connector motion/load, distributed pressure, and material properties via field variables.

Input File Usage: Use the following option to define an actuator in Abaqus:

**AMPLITUDE*, NAME=*name*, DEFINITION=ACTUATOR

There are no data lines associated with this amplitude definition to define the amplitude value at a given simulation time. Instead, the amplitude value at a given simulation time is imported from the FMU. The user-specified name can be referenced in any of the Abaqus options that can refer to an amplitude (such as **CLOAD*, **BOUNDARY*, **FIELD*, and **CHANGE FRICTION*). Typically, the magnitude of the load or boundary condition should be set to 1.0 so the total magnitude of the signal is applied. This is especially important when the coupling is bidirectional, such as a closed-looped control system.

For example, use the following options to define a force actuator in the 3-direction:

**AMPLITUDE*, NAME=*name*, DEFINITION=ACTUATOR
**CLOAD*, AMPLITUDE=*name*
101, 3, 1.0

Mapping Signal Names between Abaqus and FMUs

The mapping between actuators and sensors in the Abaqus analysis and the signals in the FMUs is defined in the SIMULIA Co-Simulation Services configuration file. The configuration file defines the simulation properties of the system and the numerical methods employed to simulate the system. It is recommended that you modify an existing configuration file from an example problem to adapt it to your system.

Executing the Coupled Analysis

You execute the Abaqus job as described in *SIMULIA Co-Simulation Services FMU Execution*.

Field Mapper

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Sequentially Coupled Multiphysics Analyses*
- *Co-Simulation*
- **FIELD MAPPER CONTROLS*
- **EXTERNAL FIELD*
- **CO-SIMULATION*

Overview

Several capabilities in Abaqus and other SIMULIA products use the field mapper to provide spatial mapping of fields between domains having different mesh topologies.

The field mapper:

- provides mapping between point, surface, and volume domains;
- provides point-cloud mapping to surface and volume domains;
- uses various search and mapping algorithms;
- supports scalar, vector, tensor, and quaternion fields; and
- allows evaluation of fields at nodes, at surface facet centers, and on element centers or integration points.

This section provides an overview of the field mapper with a focus on Abaqus capabilities. The various capabilities in Abaqus might use only a subset of the field mapper functionality. For further information about restrictions, consult the section describing the specific Abaqus capability.

The SIMULIA Co-Simulation Services (CSS) also uses the field mapper, where the CSS Configuration file specifies the mapper settings. This section does not address the CSS Configuration file settings, for that see *Mapping Specification*); however, it describes the mapping algorithm as it applies to the SIMULIA Co-Simulation Services.

Field Mapping: General Remarks

A field is a physics quantity that spans a numerical domain. The numerical domain can be a set of discrete points associated with a body, a surface on a body, or the volume of a body. In Abaqus discrete points are represented by a node set, a surface domain is represented by a surface definition, and a volume domain is represented by an element set. The field mapper maps fields between domains with dissimilar mesh topologies.

The field mapper supports mapping between similar domain types (point-to-point, surface-to-surface, and volume-to-volume) and mapping between points-to-surfaces and points-to-volume. Mapping involving points limits the mapping algorithms that you can use because no underlying region topology exists and insufficient information is available to accurately interpolate and integrate the fields.

A field is categorized as either extensive or intensive based on the physics properties. For extensive fields, the amount of matter within the body determines the field values. Typically, these are integrated values over the surface or volume of the body. Examples of extensive fields include forces, heat fluxes, and masses, which define a total quantity. For intensive fields, the amount of matter within the body does not influence the field values. Examples of intensive fields include pressure, surface heat flux density, and mass density, which are quantities defined per unit area or volume. Displacement, velocity, and temperature are also examples of intensive fields

because these fields do not depend on the amount of matter within the body. The field mapper provides conservative integration techniques for mapping intensive to extensive fields; for example, mapping of a traction field to nodal forces.

Fields can be evaluated at nodes, at the center of surface facets, or at the center or integration points of elements. It is best to evaluate fields where they are computed by the solver because they are most accurate at those locations. For example, stresses in a finite element analysis are most accurate at the integration points where they are computed, while stresses can also exist at nodes for output purposes. However, the stresses at the nodes are extrapolated quantities, and they might subsequently be averaged between adjacent elements before writing output to a results file. For co-simulation, Abaqus always selects the appropriate location to evaluate fields. In a sequential analysis, you must ensure that fields are written at the appropriate locations to the results file in the previous analysis.

Domain Coupling for Multiphysics and Multiscale Simulations

When coupling physics to solve a multiphysics or multiscale simulation, you must consider whether the numerical domains overlap or abut to define the domain coupling type. Domain coupling types include point coupling, surface coupling, and volume coupling.

- Point coupling is coupling that occurs at discrete locations associated with a body. An example of point coupling is coupling Abaqus with a one-dimensional (1D) reduced-order model. In this case, the points of the 1D model must correspond with the co-located nodes in the Abaqus model.
- Surface coupling involves distinct domains where fields are exchanged and mapped across a common interface. An example of surface coupling is a fluid-structure simulation, where the fluid and solid domains are distinct, and fields are exchanged and mapped across the interface surface where the fluid and solid adjoin.
- Volume coupling involves domains that overlap and fields that are exchanged and mapped over the volume of a body. An example of volume coupling is a sequential thermal-stress simulation; the temperature field within the body causes thermal expansion, and the thermal and stress domains overlap.

Select the domain coupling type (point, surface, or volume) independent of where the field is evaluated. For example, in a sequential thermal-stress analysis, you should define the volume of the body by providing an element set, although temperature is evaluated at nodes. By specifying a volume, the mapper uses the topology information to provide accurate mapping. If you select points to represent the domain, the mapper uses a less accurate point-cloud algorithm.

Mapping Operations

Mapping operations consist of two parts:

- a search where an association is made between members (nodes, elements, or integration points) of the source and target meshes; and
- a calculation where field values are transferred from the source to the target meshes using interpolation and integration algorithms.

The search algorithm establishes a one-to-one association between members of the source and target meshes. How such an association is formed and whether members of interest are searched on the source or target mesh depends on the mapping algorithm (see *Mapping Algorithms*). When coupling surface domains, the surface normal is used to distinguish which faces (SPOS or SNEG) are abutting. The search is performed such that the surface normals of the source and target regions are facing each other.

For discussion purposes, a mapping category defines a set of similar mapping operations performed when mapping a field from the source mesh to the target mesh. The field's physics properties (intensive or extensive) and the location at which it is evaluated define the category (see *Table 1*). Within a mapping category, there might be one or more algorithms from which you can select.

Table 1: Mapping categories.

Category	Source Field			Target Field		
	Nodal		Integration Point	Nodal		Integration Point
	Intensive	Extensive	Intensive	Intensive	Extensive	Intensive
1	√			√		√
2		√			√	
3			√	√		√
4			√		√	

Mapping Algorithms

Four mapping algorithm categories are described below. To simplify the discussion, the term "element" represents an element for a volume domain and a surface facet for a surface domain.

Category 1: Interpolation Based on the Source Element's Shape Function

Mapping category 1 is applicable when mapping intensive nodal fields from the source mesh to either intensive nodal or integration-point fields on the target mesh. Examples include mapping nodal displacements or mapping velocity from a structural solver (computed at nodes) to velocity of a CFD solver (typically computed at the fluid cell center) between source and target meshes.

If the source mesh is element based, the target node or integration point is projected onto the source mesh, and the search algorithm finds the closest source element and the location within the source element where the target point is projected. The target field is then interpolated using the source element's shape function. If the source mesh is discrete, a similar algorithm is employed using a point-cloud search algorithm. The point-cloud search algorithm find the closest source nodes and forms a virtual element to interpolate the field values.

Category 2: Interpolation Based on the Target Element's Shape Function

Mapping category 2 is applicable when mapping extensive nodal fields to extensive fields; for example, mapping nodal forces between source and target meshes.

If the source mesh is element based, the source node is projected onto the target mesh, and the search algorithm finds the closest target element and the location within the target element where the source node is projected. The source field is then distributed using the target element's shape function. If the target mesh is discrete, a similar algorithm is employed using a point-cloud search algorithm. The point-cloud search algorithm finds the closest target nodes and forms a virtual element to distribute the field values. In this case, the total quantity is conserved across the interface; however, the distribution might be less accurate.

Category 3: Extrapolation Followed by Interpolation on the Source Element

Mapping category 3 is applicable when mapping intensive integration-point fields to either intensive nodal or integration-point fields on the target mesh. An example is mapping stresses between source and target meshes.

The source mesh must be element based. The field is first extrapolated to each element's nodes on the source mesh. Following the extrapolation, the elemental stress values are retained and averaging is done among neighboring elements. The target node or integration point is projected onto the source mesh, and the search

algorithm finds the closest source element and the location within the source element where the target point is projected. The target field is then interpolated using the source element's shape function.

Category 4: Integration of Intensive to Extensive Fields

Mapping category 4 uses integration methods and is applicable when mapping intensive integration-point fields to extensive nodal fields. Two algorithms are available: the subelement algorithm and the global conservative algorithm.

Subelement Algorithm

The subelement algorithm provides conservative mapping with a smooth distribution. It requires that the target mesh is element based. Each target element is divided into subelements. For each subelement, this algorithm locates the source field values according to the procedure described in *Category 1: Interpolation Based on the Source Element's Shape Function* and sums the values over all the subelements. The number of subelements is determined automatically to provide accurate results.

Input File Usage: Use the following option to select the subelement algorithm:

**FIELD MAPPER CONTROLS*, MAPPER ALGORITHM=SUBELEMENT

Global Conservative Algorithm

The global conservative algorithm provides conservative results over the entire region. It requires that the target mesh is element based. This algorithm is computationally less expensive than the subelements algorithm; however, the mapped field might show an uneven distribution in some cases. The global conservative algorithm makes the source field extensive by integrating the field over the source elements and then, for every source node, computing field values according to the procedure described in *Category 2: Interpolation Based on the Target Element's Shape Function*.

Input File Usage: Use the following option to select the global conservative algorithm:

**FIELD MAPPER CONTROLS*, MAPPER ALGORITHM=GLOBAL CONSERVATIVE

Modifying the Search Position Tolerance Distance

Because the source and target meshes need not be coincident, the search algorithm must determine whether or not to map a member of the target mesh. The distance used to make that determination is referred to as the search position tolerance distance. The default position tolerance is 50% of the average element diagonal length in the domain. In certain circumstances, you might want to modify the search position tolerance distances; for example, if your model has complex geometry or if the mesh topology in one dimension is significantly different in other dimensions.

You can specify four position tolerances.

- Define a search position tolerance distance in the positive surface normal direction for surface domains; only points within the position tolerance normal to the surface are considered.
- Define a search position tolerance distance in the negative surface normal direction for surface domains or a search radius for the point-cloud algorithm for point domains.
- Define a position tolerance for how far a point can be located outside an element for volume coupling or outside a surface facet in the plane of the surface for surface coupling.
- Define a search tolerance radius for the unmapped members treatment (see *Nearest Neighbor Method*).

You can provide the position tolerance distance as an absolute distance or a relative distance based on the characteristic region dimension. Absolute tolerances might be more appropriate if the mesh discretization varies significantly, such that the characteristic dimension is a poor representation in certain parts of the mesh.

Input File Usage: Use the following option to specify absolute search tolerance distances:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, SEARCH TOLERANCE=ABSOLUTE
positive normal distance, *negative normal distance or point-cloud radius*, *distance outside element*, *nearest neighbor radius*

Use the following option to specify a relative search tolerance for search criteria. In this case, the relative values are multiplied by the characteristic domain dimension.

**FIELD MAPPER CONTROLS*, NAME=*controls name*, SEARCH TOLERANCE=RELATIVE
positive normal distance tolerance, *negative normal distance tolerance or point-cloud radius tolerance*, *distance tolerance outside element*, *nearest neighbor radius tolerance*

Selecting the Base Configuration

Typically, the search is based on the original nodal coordinates of the source mesh. You might want to use the deformed configuration of the source mesh; for example, in a case where the model is remeshed in the subsequent analysis using the deformed shape of the first analysis.

Input File Usage: Use the following option to use the original configuration of the source mesh:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, CONFIGURATION=REFERENCE

Use the following option to use the deformed configuration of the source mesh:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, CONFIGURATION=DEFORMED

Unmapped Members Treatment

If the search fails to make an association, the field mapper reports unmapped members. For most Abaqus features, the simulation ends, and the unmapped members are reported. Common causes of such failures include:

- The source and target domains use different unit systems.
- The source and target regions are not co-located.
- The target domain is larger than the source domain, requiring extrapolation.
- The search tolerance distances are inadequate for complex geometry or poor mesh topology.

You can try to correct the mesh or adjust the search tolerance distances. If these attempts fail, you can specify how the field mapper treats the unmapped members. Treatment is intended for circumstances where you have only a few unmapped members. You should interpret results carefully in the vicinity of the unmapped members. There are two treatment methods available: the nearest neighbor method and the constant value method.

Nearest Neighbor Method

The field mapper can approximate values by assigning a value from a neighboring node or element. The search for candidate neighbors is limited by the nearest neighbor radius tolerance (see *Modifying the Search Position Tolerance Distance*).

Input File Usage: Use the following option to assign values from a neighboring node or element:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, UNMAPPED MEMBERS TREATMENT=NEAREST NEIGHBOR

Constant Value Method

Alternatively, you can specify a constant value to assign to unmapped members. For example, when transmitting a heat flux across a surface, you can assign a zero heat flux at an unmapped member. In such a case, the total heat flux is still conserved across the entire interface, but, locally, a zero heat flux is used for the unmapped member.

Input File Usage: Use the following option to assign constant values to unmapped members:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, UNMAPPED MEMBERS TREATMENT=CONSTANT VALUE
…
second data line to define constant values for unmapped members

For example, you can use the following option to assign a zero heat flux at an unmapped member:

**FIELD MAPPER CONTROLS*, NAME=*controls name*, UNMAPPED MEMBERS TREATMENT=CONSTANT VALUE
…
0.0

Specifying Tensor Averaging

When mapping tensor quantities, you can specify how to use the averaging technique. You can average the tensor component-by-component, using a technique that preserves the eigenvalues of the tensor, or using a technique that preserves the von Mises invariant.

Input File Usage: Use the following option to use the component-by-component technique:

**FIELD MAPPER CONTROLS*, TENSOR AVERAGING=COMPONENT

Use the following option to preserve the eigenvalue:

**FIELD MAPPER CONTROLS*, TENSOR AVERAGING=EIGENVALUE

Use the following option to preserve the von Mises invariant:

**FIELD MAPPER CONTROLS*, TENSOR AVERAGING=INVARIANT

Design Sensitivity Analysis

In this section:

- *Design Sensitivity Analysis Overview*
- *Direct Design Sensitivity Analysis*
- *Adjoint Design Sensitivity Analysis*

Design Sensitivity Analysis Overview

Overview

Design sensitivity analysis includes various methods for obtaining the sensitivities applied in optimization approaches and algorithms.

A design response refers to a physical quantity of interest; for example, the displacement at a specific point or the maximum stress for a part that may be one of the key performance indicators for the part during its application. During the design of the part, you may be interested in optimizing (minimizing, maximizing, or setting constraints on) one or more design responses. A design variable, on the other hand, refers to a physical quantity (such as the thickness of shells for various regions of a shell structure or the radius of individual beams in a lattice structure) that you can modify to ensure optimum values for one or more design responses. A design sensitivity refers to the derivative of a design response with respect to a design variable. Therefore, a design sensitivity provides guidance for modifying a design variable with the objective of improving a design response, thereby improving the service performance of a part. In other words, a design sensitivity tells you in which "direction" and by "how much" you should modify the design variable (for example, should a shell thickness be decreased or increased and by what amount?) to obtain improved performance.

Figure 1 illustrates two examples of how sensitivities guide modifications of design variables to improve the structural performance of stress and eigenfrequency (design) responses, respectively, of a part. As the figure shows, the sensitivities represent linearized first-order derivatives at a given value of the design variable. Typically, sensitivities are used to iteratively modify a given design configuration to improve the structural performance. Such design modifications can be done manually or in an automated manner using optimization algorithms based upon mathematical programming. The focus here is not on the optimization algorithms but on various methods for obtaining the sensitivities. You can find examples of various optimization algorithms in the Tosca or Isight documentation.

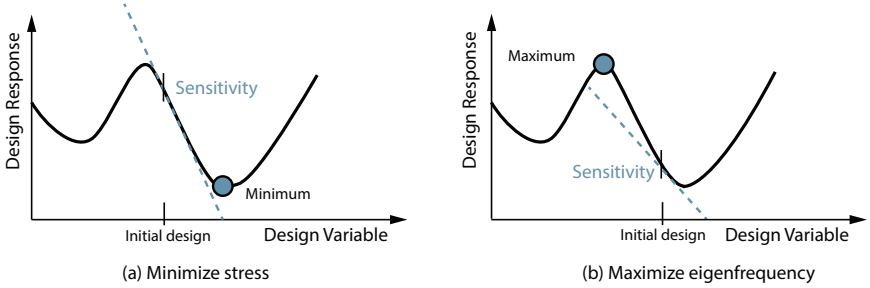


Figure 1: Sensitivities guide modification of the design variable.

Practically, the approaches for computing design sensitivities can be classified as finite difference–based sensitivity approaches and analytical and semi-analytical sensitivity approaches. These approaches are further classified as follows:

Finite difference–based sensitivity approaches

- Forward finite difference.
- Backward finite difference.
- Central finite difference.

Analytical and semi-analytical sensitivity approaches

- Direct approach.
- Adjoint approach.

Finite difference approaches are very general methods for estimating the sensitivities. Although the finite difference–based sensitivities are often not particularly accurate, they can be computed very efficiently for finite element models having a small number of design variables (< 5–10). These methods are also easy to implement. Typically, sensitivities computed using finite difference methods are applied in Isight-based optimization workflows.

The analytical and semi-analytical design sensitivity analysis approaches in Abaqus are designed for the class of problems having a medium number (> 5 but < 50–1000) to a large number (>> 1,000–1.0×10^8) of either design variables or design responses. Direct sensitivity analysis is recommended for problems having a medium to large number of design responses but a small number of design variables (see *Direct Design Sensitivity Analysis*). Adjoint sensitivity analysis is recommended for problems having a large number of design variables but a small to medium number of design responses (see *Adjoint Design Sensitivity Analysis Overview*).

For the class of problems that involve a medium to a large number of both design variables and design responses, no general solution for computing design sensitivities in an efficient manner exists today. However, for such problems, the number of design responses can sometimes be significantly reduced by replacing a large group of design responses of a certain type with the maximum, the minimum, or the average value over that group.

References:

- *Direct Design Sensitivity Analysis*
- *Adjoint Design Sensitivity Analysis Overview*

Direct Design Sensitivity Analysis

Products: Abaqus/Design

References:

- *Parametric Modeling*
- *Parametric Shape Variation*
- **STEP*
- **DESIGN PARAMETER*
- **DESIGN RESPONSE*

Overview

Direct design sensitivity analysis (DSA):

- is performed with Abaqus/Design, an add-on option for Abaqus/Standard;
- provides the sensitivities of responses with respect to specified design parameters;
- is available for static stress and frequency analysis using models that have only stress/displacement elements; and
- can include design parameters affecting: material properties (elastic, hyperelastic, and hyperfoam models); section properties; concentrated forces and moments; and nodal coordinates (and beam and shell normals if applicable).

Abaqus/Standard also computes design sensitivities for a class of design responses using the adjoint method. See *Adjoint Design Sensitivity Analysis Overview* for a detailed discussion of the adjoint method for computing design sensitivities and the design responses and design variables that are supported.

Design Sensitivity Analysis

The design sensitivity analysis (DSA) capability provides the derivatives of certain output variables with respect to specified design parameters. These derivatives are commonly referred to as *sensitivities*, because they provide a first-order measure of how sensitive the output variable is to a change in the design parameter. The output variables for which sensitivities are computed are called *design responses* or simply *responses*. Design parameters are chosen from a set of existing analysis parameters. As an example, you can choose to obtain the derivatives of stresses with respect to Young's modulus; stress is the response, and Young's modulus is the design parameter. The sensitivities are computed based on the direct differentiation method used in conjunction with the semi-analytical computational technique. In the semi-analytical technique some derivatives are computed using numerical (finite) differencing, thus requiring perturbations of the design parameters. For these derivatives by default Abaqus/Design will use a central differencing scheme and automatically determine appropriate perturbation sizes based on a heuristic algorithm. You can override these defaults by specifying the numerical differencing method and the perturbation sizes directly. A full discussion of DSA theory is given in *About design sensitivity analysis*.

Activating DSA

You activate DSA on a step-by-step basis.

Input File Usage: Use the following option to activate DSA in a particular step:

**STEP*, DSA=YES

Activating DSA in Multiple Steps

Once DSA is activated in a general step, it remains active in all subsequent general steps until it is deactivated in a subsequent general step. Once DSA is activated in a perturbation step, it remains active in all subsequent consecutive perturbation steps until it is deactivated in a subsequent consecutive perturbation step. However, if DSA is activated in a step whose procedure is not supported for DSA, DSA will be deactivated until it is activated again.

Input File Usage: Use the following option to deactivate DSA in a particular step:

**STEP*, DSA=NO

Specifying Design Parameters

You can define multiple parameters to be used in place of Abaqus input quantities for an analysis. You must indicate which of these parameters are to be considered as design parameters.

Input File Usage: Use the following option to define analysis parameters:

**PARAMETER*
par1=x
par2=y
. . .

Use the following option to specify the design parameters:

**DESIGN PARAMETER*
par1, *par2*, . . .

Restrictions on Design Parameters

The following are restrictions on design parameters:

- Design parameters can be associated only with floating point data. The following analysis components can include design-dependent data:
  - Beam sections integrated during analysis (*Using a Beam Section Integrated during the Analysis to Define the Section Behavior*)
  - Concentrated loads (*Concentrated Loads*)
  - Elastic materials (*Linear Elastic Behavior*)
  - Friction (*Frictional Behavior*)
  - Gasket sections (*About Gasket Elements*)
  - Hyperelastic materials (*Hyperelastic Behavior of Rubberlike Materials*)
  - Hyperfoam materials (*Hyperelastic Behavior in Elastomeric Foams*)
  - Membrane sections (*Membrane Elements*)
  - Local orientations (*Orientations*)
  - Shell sections integrated during analysis (*Using a Shell Section Integrated during the Analysis to Define the Section Behavior*)
  - Solid sections (*Solid (Continuum) Elements*)
  - Transverse shear stiffnesses (*Choosing a Beam Element* or *Shell Section Behavior*)

- Shape design parameters (i.e., design parameters that affect nodal coordinates and beam and/or shell normals) can be used only in conjunction with parametric shape variations (see *Parametric Shape Variation*).
- Design parameters must be mutually independent.
- Design parameters cannot be tabularly dependent (see *Parametric Modeling*).

Specifying Responses

Response requests are specified using a syntax analogous to that for specifying output requests to the output database. Except for eigenvalues and eigenfrequencies, there are no default responses—if no responses are requested, no response sensitivities will be output. If DSA is active in a frequency step, eigenvalue and eigenfrequency sensitivities will be output automatically. Specifying a response will cause output of both the response and the response sensitivities.

Input File Usage: Use the following options to request design responses:

**DESIGN RESPONSE*, FREQUENCY=*interval*, MODE LIST
**CONTACT RESPONSE*, MAIN=*main name*, NSET=*nset name*, SECONDARY=*secondary name*
**ELEMENT RESPONSE*, ELSET=*elset name*
**NODE RESPONSE*, NSET=*nset name*

Requesting Responses in Multiple Steps

Unless respecified, response requests defined in a step propagate to subsequent steps according to the following rules:

1. Requests in general steps propagate to subsequent general steps.
2. Requests in linear perturbation steps propagate to subsequent consecutive linear perturbation steps.
3. When a non-DSA step appears between DSA steps, the responses must be respecified in the DSA step following the non-DSA step.

Restrictions on Responses

The available responses are a subset of the existing output variables. The valid responses based on procedure type are described below.

- For static steps the valid responses are:
  - Node responses: U and RF
  - Element responses: S, SF, SINV, SP, E, SE, EP, EE, EEP, LE, LEP, NE, NEP, ENER, ELEN, EVOL, and MASS
  - Contact responses: CSTRESS and CDISP
- For frequency steps the valid responses are:
  - Node responses: None
  - Element responses: MASS
  - Contact responses: None
  - Eigenvalue (EIGVAL) and eigenfrequency (EIGFREQ) sensitivities are output automatically.

Specifying Design Gradients of Design-Dependent Input Data

The DSA calculations require the gradients of the design-dependent input data with respect to the design parameters. For example, if Poisson's ratio, ν, is made dependent on a design parameter, say h, the gradient $d\nu/dh$ is required. Design gradients with respect to shape design parameters are specified differently than those with respect to other design parameters.

Specifying Design Gradients with Respect to Shape Design Parameters

Gradients with respect to shape design parameters must be specified using a parametric shape variation definition (see *Parametric Shape Variation*). For the purposes of DSA if the parameter to which the shape variation data refer is a design parameter, the shape variation data are interpreted as the gradients of the nodal coordinates with respect to the design parameter. If a nonzero value is given for the shape parameter, Abaqus/Design will also perturb the base coordinates.

Input File Usage: Use the following option to specify the design gradients for shape design parameters:

**PARAMETER SHAPE VARIATION*, PARAMETER=*design parameter*

Specifying Gradients for Non-Shape Design Parameters

For non-shape design parameters, by default Abaqus/Design will use numerical differentiation to calculate design gradients based on the information you provide. However, you can override this default behavior by specifying the gradients directly using Python expressions (see *Parametric Modeling*). You specify a design parameter as the independent parameter and a list of the parameters that depend on that design parameter. Only one independent (design) parameter can be given for each design gradient definition.

Input File Usage: Use the following option to specify the design gradients for non-shape design parameters:

**DESIGN GRADIENT*, INDEPENDENT=*design parameter*,
DEPENDENT=*(list of dependent parameters)*

History Dependence and Formulation Type in a Multi-Increment Analysis

Both total and incremental formulations are implemented for DSA. The choice of formulation depends on whether or not an analysis is history dependent. Below is a brief description of these formulation types. A more detailed discussion can be found in *About design sensitivity analysis*. By default, the incremental DSA formulation is used. You can specify the DSA formulation only for the entire model; this specification is ignored if given as part of a step definition.

Incremental DSA Formulation

In the incremental formulation the problem is assumed to be history dependent. Abaqus/Design solves for the incremental displacement sensitivities, and the total displacement sensitivity is updated at the end of the increment. Due to the history dependence, the incremental displacement sensitivities for the current increment depend on the sensitivities of the state variables at the beginning of the increment, in the same sense that incremental displacements depend on the state variables at the beginning of the increment for equilibrium analyses. Thus, Abaqus/Design must also compute and update state variable sensitivities in each increment. Consequently, DSA must be activated for *all* steps up to the last step in which DSA is active, and the DSA calculations will be done at *all* increments in these steps, regardless of whether or not a design response is requested for a given step. If a response is requested for a step, the specified response frequency is ignored for the purposes of the DSA

calculations (the frequency at which the output is written will still be governed by the specified response frequency).

The disadvantage of the incremental DSA formulation is its cost, due to the necessity of computing both state variable and incremental displacement sensitivities at each increment prior to the last DSA increment. This increased cost is unavoidable if the problem is history dependent but is unnecessary if the problem is history independent. Thus, the total DSA formulation should be chosen for problems that are not history dependent.

Input File Usage: **DSA CONTROLS*, FORMULATION=INCREMENTAL

Total DSA Formulation

In the total displacement formulation the total displacement sensitivities are calculated directly based on the assumption that the problem is not history dependent. In other words, the displacement sensitivities do not depend on sensitivity results calculated in previous increments. Thus, the advantage of the total formulation is that the sensitivity calculations need only be done at increments of interest. You can control when DSA calculations are done by activating DSA for only the desired steps and specifying the desired frequency for each design response request.

You may choose to use the total DSA formulation in problems that are known to be history dependent. However, in this case the DSA solution is approximate, with the degree of approximation increasing as the problem becomes more strongly history dependent. To assess the validity of using the total DSA formulation, it is recommended that you run both an incremental and total sensitivity analysis for a typical problem and compare the results.

Input File Usage: **DSA CONTROLS*, FORMULATION=TOTAL

DSA in Linear Perturbation Steps

The sensitivity of the perturbation response can be calculated in a linear perturbation step (see *General and Perturbation Procedures*). The perturbation response will include the effects of stress and load stiffening in the base state if geometric nonlinearity is considered. Since we need to calculate the sensitivity of an incremental (perturbation) response, the sensitivity of the stress and load stiffening effects must be known at the end of the base step. Thus, if geometric nonlinearity is considered in the base step, DSA must also be active in the base step, irrespective of the type of formulation (total or incremental).

Determination of Design Parameter Perturbation Sizes

The basis of the semi-analytic technique is the use of numerical differencing to obtain derivatives of certain element vectors and matrices (see *About design sensitivity analysis*). Abaqus/Design will automatically determine appropriate perturbation sizes to be used in the semi-analytic technique unless you specify them directly. Abaqus/Design determines the perturbation sizes using a heuristic perturbation sizing algorithm based on the behavior of a scalar s associated with an element. By default, the perturbation sizing algorithm is applied only for the first increment (static procedure) or first mode (frequency procedure) in each step for which DSA is active. The perturbation sizes are then reused for the remaining increments or modes in the step for which DSA calculations are done.

The goal of the algorithm is to find perturbation sizes that are optimal for numerical differencing. Differencing formulas are based on Taylor series expansions, and the order of approximation of the derivative to be computed is reflected in the terms that are neglected in the series. The accuracy of the approximated derivatives often depends strongly on the perturbation size used in the differencing formula. Choosing a perturbation size that is too large will cause a truncation error, which occurs when the order of approximation is no longer valid (i.e., as a result of truncating higher-order terms in the Taylor series). A perturbation size that is too small will lead to inaccuracies in the differencing operations due to round-off, typically referred to as a cancellation error.

The algorithm attempts to find perturbation sizes giving the best compromise between cancellation and truncation errors by observing the behavior of *s*. For each design parameter *s* is computed for perturbation sizes spanning several orders of magnitude. The error in *s* between consecutive perturbation sizes is calculated as

$e_i = |s_i - s_{i-1}| / s_i$. The perturbation size yielding an acceptable error, e_a, is chosen as the best perturbation size.

This scalar *s* is selected as follows:

- Static procedure. For static steps *s* is chosen as the norm of the element pseudoload (the partial derivative of the element residual with respect to the design parameters).
- Frequency procedure. For frequency steps *s* is computed from the element contribution to a matrix involving the derivatives of the mass and stiffness matrices (namely $\left(\frac{DK^{NM}}{Dh} - \lambda\frac{DM^{NM}}{Dh}\right)$, where K^{NM} is the stiffness, M^{NM} is the mass, *h* is the design parameter, and λ is an eigenvalue). The scalar *s* is taken as the projection of this matrix onto an eigenvector $\overline{\phi}^N$. If the perturbation sizing algorithm is applied to a mode with a distinct eigenvalue, $\overline{\phi}^N$ is taken as the eigenvector associated with this mode. However, if a mode happens to be associated with a repeated eigenvalue, $\overline{\phi}^N$ is taken as the sum of all the eigenvectors associated with the repeated eigenvalue. Thus, the entire set of modes associated with a repeated eigenvalue will be treated simultaneously by the perturbation sizing algorithm (the eigenvalue sensitivities of a repeated eigenvalue are obtained simultaneously from the same reduced eigenvalue system).

See *About design sensitivity analysis* for further details on the selection of *s*.

Controlling the Numerical Differencing Behavior

You can control various aspects of the numerical differencing operations. These aspects are described in detail in the following sections. You can specify DSA controls for the entire model and/or for individual steps. Specifying these controls for the entire model has the effect of creating new default values for the various settings. When you specify these controls for individual steps, the following propagation rules are enforced:

- Once DSA controls are specified in a non-perturbation step, they remain in effect for all subsequent non-perturbation steps, unless they are respecified or reset.
- Once DSA controls are specified in a perturbation step, they remain in effect for all subsequent consecutive perturbation steps, unless they are respecified or reset.

Resetting DSA Controls

You can reset DSA controls only for individual steps. If DSA controls are specified for the entire model, resetting them in a particular step will reset the numerical differencing behavior to the behavior specified for the entire model; otherwise, the behavior will be reset to the original default values. Any additional changes specified will be applied after the behavior is reset.

Input File Usage: Use the following option to reset the DSA controls for a particular step:

**DSA CONTROLS*, RESET

Changing the Defaults for the Heuristic Perturbation Sizing Algorithm

The following two sections describe how certain parameters associated with the perturbation sizing algorithm can be changed from their default values for purposes of computational efficiency and accuracy.

Changing the Default Tolerance

By default, the tolerance ϵ_a is set to 1.0×10^{-4}. Warning messages are written to the message file for elements for which this tolerance is not achieved. These elements are collected in element sets and can be viewed in the Visualization module of Abaqus/CAE. It is important to understand that this tolerance controls the effort expended in obtaining an optimum perturbation size; it is not a direct measure of the accuracy of the numerical differentiation.

Input File Usage: Use the following option to override the default tolerance:

**DSA CONTROLS*, TOLERANCE=*tolerance*

Changing the Frequency at Which the Perturbation Sizing Algorithm Is Used

Determining perturbation sizes using the heuristic algorithm is computationally intensive. You can specify the frequency at which the perturbation sizes are recalculated. For example, specifying a sizing frequency of *n* will cause Abaqus/Design to determine new perturbation sizes at every *n* increments or eigenmodes. The perturbation size will always be recalculated at the first increment or eigenmode in each step for which DSA is active, which is equivalent to specifiying a sizing frequency of 0. Since the perturbation sizing algorithm is computationally intensive, care should be exercised to ensure that the sizing frequency is as large as possible (or zero).

As discussed above, the perturbation sizing algorithm is applied simultaneously to all modes associated with a repeated eigenvalue. Thus, the actual number of modes associated with a repeated eigenvalue that are "hit" based on the sizing frequency is irrelevant, so long as it is at least one.

Input File Usage: Use the following option to specify the frequency at which the perturbation sizes are recalculated:

**DSA CONTROLS*, SIZING FREQUENCY=*frequency*

Overriding the Default Heuristic Perturbation Sizing Algorithm

If an appropriate perturbation size is already known for a particular design parameter (from previous analyses of similar problems, for example), economy can be gained by applying this perturbation size directly rather than having Abaqus/Design automatically find the perturbation size. You can specify either forward differencing or central differencing directly together with an absolute perturbation size for each design parameter. If you override the default algorithm, it is up to you to choose perturbation sizes that will lead to accurate sensitivities.

Input File Usage: Use the following option to override the default heuristic perturbation sizing algorithm for a given design parameter:

**DSA CONTROLS*
design parameter, FD (forward differencing) or CD (central differencing), *absolute perturbation size*

For example, to specify an absolute perturbation size of 0.001 and forward differencing for design parameter `despar` use the following input:

**DSA CONTROLS*
despar, FD, 0.001

This data line is specified for each design parameter for which the default scheme is to be overridden.

Accuracy of the DSA Solution

As can be seen in *About design sensitivity analysis*, the accuracy of the DSA solution is dictated by both the accuracy of the numerically computed derivatives and, for nonlinear static analysis, the accuracy of the tangent stiffness matrix. The accuracy of the numerically computed derivatives is governed by the semi-analytic DSA algorithm; you can control it by specifying DSA controls. In nonlinear static analysis DSA uses the tangent stiffness matrix formed during the last equilibrium iteration. It is possible that the accuracy of the tangent stiffness matrix needed to achieve an accurate equilibrium solution may be insufficient to achieve an accurate DSA solution. In such cases you can tighten the convergence tolerances during the equilibrium analysis so that a more accurate tangent stiffness matrix is obtained (see *Commonly Used Control Parameters*). Furthermore, an accurate equilibrium solution often can be obtained when unsymmetric terms in the tangent stiffness are ignored (i.e., the unsymmetric matrix storage and solution scheme is not used; see *Defining an Analysis*). However, even if mildly unsymmetric stiffness terms are neglected, the DSA solution may be inaccurate. Therefore, it is recommended that the unsymmetric solution scheme be used for DSA when the tangent stiffness matrix is known to be unsymmetric.

In some cases a response at a certain instant in time may be discontinuous with respect to a design parameter. For example, at this point of discontinuity a variation in the design parameter may cause a node to come into contact, frictional behavior to change from sticking to sliding, or a material point to transition from elastic to inelastic behavior. Since the DSA calculations make use of numerical differencing, it is possible that the perturbation of the design parameter used in the differencing scheme may result in values of the response to be differenced that lie on opposite sides of the discontinuity. If this occurs, the accuracy of the computed derivative cannot be guaranteed. Mathematically speaking, the derivative (sensitivity) of the response with respect to the design parameter does not exist at the point of discontinuity. Practically speaking, it is unlikely that the response at any given instant will lie precisely on the discontinuity. In cases where the response is near a discontinuity, if you choose to use the default perturbation sizing algorithm, the algorithm will attempt to choose design parameter perturbation sizes such that the values of the perturbed responses remain on the same side of the discontinuity. In addition, for contact elements DSA calculations are not performed in increments in which the associated contact node is open. Typically, the global results in any increment are not affected by a few discontinuous points in the model.

Design Dependence and Supported Features

Responses depend on design parameters *explicitly* and *implicitly*. Implicit design dependence is the dependence on the design parameter through the solution variables; therefore, this type of dependence can be quantified only after the DSA solution is obtained (recall that the DSA solution is the total displacement sensitivity for the total formulation and the incremental displacement sensitivity for the incremental formulation). All other design dependencies are explicit, meaning that they can be resolved without knowing the DSA solution. The types of dependencies can be identified by looking at the form of the sensitivity of a response, say r, with respect to a design parameter, say h. This sensitivity is expressed as

$$\frac{dr}{dh} = \frac{\partial r}{\partial h} + \frac{\partial r}{\partial u^N}\frac{du^N}{dh}$$

for the total formulation and

$$\frac{dr}{dh} = \frac{\partial r}{\partial h} + \frac{\partial r}{\partial \alpha^t}\frac{d\alpha^t}{dh} + \frac{\partial r}{\partial \Delta u^N}\frac{d\Delta u^N}{dh}$$

for the incremental formulation, where u^N is a displacement degree of freedom and α^t represents state variables at the beginning of the increment (see *About design sensitivity analysis* for further details). The state variables include the displacements at the beginning of the increment. In both cases the last term on the right-hand side represents the implicit design dependence through the solution variables.

It is observed from the incremental equation above that the explicit design dependence consists of two terms.

The first of these, $\frac{\partial r}{\partial h}$, represents a *direct design dependence*, because this term arises from the direct dependence of the response on the design parameter. The second explicit term, $\frac{\partial r}{\partial \alpha^t}\frac{d\alpha^t}{dh}$, represents the dependence on the design parameter through the state variables at the beginning of the increment. For the total formulation, it is seen that the explicit term involves only direct design dependence.

Any feature for which direct design dependence calculations are implemented in Abaqus will be referred to as *supported* for DSA. Supported and unsupported features can be mixed in an analysis, unless the supported features cause unsupported features to become directly design dependent (an example of this would be making the Young's modulus for a frame element design dependent, since frames are not supported for DSA).

To make a clearer distinction between the types of design dependencies, consider the more concrete example of a linear elastic truss element, fixed at one end and pulled with a concentrated load at the other end. Let $u = u^t + \Delta u$ represent the displacement at the free end, E represent Young's modulus, and L represent the length of the truss. Consider the axial stress S_{11} as the response. Although it is clear in this simple example that the stress can be computed easily as the load divided by the cross-sectional area, the finite element analysis computes the stress equivalently as $S_{11} = Eu/L$. Choosing Young's modulus, E, as the design parameter, the stress sensitivity is given by

$$\frac{dS_{11}}{dE} = \frac{\partial S_{11}}{\partial E} + \frac{\partial S_{11}}{\partial u}\frac{du}{dE} = \frac{u}{L} + \frac{E}{L}\frac{du}{dE}$$

for the total formulation and

$$\frac{dS_{11}}{dE} = \frac{\partial S_{11}}{\partial E} + \frac{\partial S_{11}}{\partial u^t}\frac{du^t}{dE} + \frac{\partial S_{11}}{\partial \Delta u}\frac{d\Delta u}{dE} = \frac{u^t+\Delta u}{L} + \frac{E}{L}\frac{du^t}{dE} + \frac{E}{L}\frac{d\Delta u}{dE}$$

for the incremental formulation. This example is a valid analysis since elastic materials and truss elements are supported for DSA. Suppose now that a frame element is added, extending the length of the structure. If the frame element shares the same Young's modulus, the analysis becomes invalid since the dependency on the design parameter E causes the frame element, which is unsupported, to become directly design dependent (i.e., the term $\frac{\partial S_{11}}{\partial E}$ would need to be computed). On the other hand, if the frame uses a different modulus, say E_1 that is not a design parameter, the analysis again becomes valid, since the frame no longer depends directly on the design parameter E.

Contact Interactions

Surface-based contact between deformable and rigid surfaces with small- or finite-sliding relative surface motion including friction is supported in a design sensitivity analysis. In all the friction models only the friction coefficients (no test data input) can be made design dependent. Shape design parameters are not valid for rigid surfaces. Contact between deformable surfaces is not supported.

Restarting a Design Sensitivity Analysis

A design sensitivity analysis can be restarted (see *Restarting an Analysis*). However, DSA must have been active in the base analysis, and no design parameter or gradient data can be modified in the restart run. The restarted analysis will follow all the DSA propagation rules that are applicable to a regular analysis. For total formulation DSA, you may choose to activate or deactivate DSA in any new step that is added to the restart run. However, for the incremental formulation DSA must have been active in the step at which restart is attempted for you to continue doing DSA in the restarted analysis.

Procedures

DSA is available in the following analysis procedures:

- Frequency analysis
- Static stress analysis (including nonlinear geometric effects and contact)

The following analysis procedures and techniques are not supported:

- Static stress analysis with the Riks method
- Static perturbation analysis with LCP contact
- Substructuring
- Mesh modification or replacement
- Importing and transferring results
- Symmetric model generation and results transfer
- Contour integrals
- Cyclic symmetry in frequency procedures

Submodeling Limitations

Design sensitivity analysis can be performed in both the global model and submodel, with the limitation that the DSA solution will not be interpolated from the global model to the submodel. This means that DSA is valid in the submodel only if the global solution that is interpolated onto the boundary of the submodel can be considered independent of the design parameters chosen for the submodel sensitivity analysis.

Material Options

The following material models are supported:

- Isotropic, orthotropic, and anisotropic elasticity
- Hyperelasticity
- Hyperfoam

In these models only directly input material coefficients (not test data) can be made design dependent. If test data are specified, that material definition can be replaced by specifying the material coefficients calculated by Abaqus/Design directly. Supported and unsupported material models can be mixed in the same analysis.

Elements

Solid, truss, shell, beam, gasket, and membrane stress/displacement elements are supported. Shell elements with five degrees of freedom per node cannot be used in a total DSA formulation. Supported and unsupported elements can be mixed in the same analysis.

Output

The responses and response sensitivities (see *Specifying Responses* above) are output only to the output database (sensitivity output to the data file and results file is not supported). The names of the sensitivities are related to the names of the responses as follows:

d_{response name}_{design parameter name}

For example, if the name of the response is S and the name of the design parameter is Young, the name of the sensitivity is d_S_Young.

Input File Template

```
*HEADING
…
*PARAMETER
Python expressions defining parameters.
*DESIGN PARAMETER
List of independent parameters to be considered as design parameters.
…
*NODE, NSET=nset
Data lines to define the nodes.
*PARAMETER SHAPE VARIATION, PARAMETER=parameter
Data lines to define the gradients of coordinates with respect to the parameter.
…
*ELEMENT, TYPE=solid element type, ELSET=elset_elastic
Data lines to define the elements.
*ELEMENT, TYPE=solid element type, ELSET=elset_hyper
Data lines to define the elements.
*SOLID SECTION, ELSET=elset_elastic, MATERIAL=elastic
*SOLID SECTION, ELSET=elset_hyper, MATERIAL=hyper
*MATERIAL, NAME=elastic
*ELASTIC
Data lines to define the elastic properties.
*MATERIAL, NAME=hyper
*HYPERELASTIC
Data lines to define the hyperelastic properties.
…
*STEP,DSA
*STATIC
…
*DESIGN RESPONSE, FREQUENCY=interval
*ELEMENT RESPONSE, ELSET=elset
Data lines to specify the element response identifier keys.
*NODE RESPONSE, NSET=nset
Data lines to specify the nodal response identifier keys.
*END STEP
```

Adjoint Design Sensitivity Analysis

In this section:

- *Adjoint Design Sensitivity Analysis Overview*
- *Adjoint Sensitivity Method for Transient Dynamics Using Abaqus/Standard*
- *Adjoint Sensitivity Method for Fatigue Workflows Using Abaqus and fe-safe*
- *Activating Adjoint Sensitivity Analysis*
- *Specifying Design Variables*
- *Specifying Design Responses in a Static or an Eigenfrequency Analysis*
- *Specifying Design Responses in a Combined Abaqus Static and fe-safe Fatigue Analysis*
- *Specifying Design Responses in a Transient Dynamic Analysis*
- *Specifying Design Responses in a Direct Steady-State Dynamic Analysis*
- *Limiting Sensitivity Output by Specifying Bounds on Design Response Values*

Adjoint Design Sensitivity Analysis Overview

Products: Abaqus/Standard

References:

- *About Structural Optimization*
- **STEP*
- **DESIGN RESPONSE*
- **TABLE COLLECTION*

Overview

Adjoint design sensitivity analysis is used to compute derivatives of design responses with respect to one or more design variables.

Adjoint design sensitivity analysis:

- is useful for nonparametric structural design and optimization workflows to produce components that are lightweight, rigid, and durable;
- is supported for the following approaches to structural sensitivity analysis: elemental topology sensitivity, elemental sizing sensitivity for regular (noncomposite) shells, layer sizing and/or layer orientation sensitivity for composite shells, elemental sizing sensitivity for circular beams, elemental ABD stiffness sensitivity for shell general sections, nodal shape sensitivities for continuum elements, and nodal bead sensitivities for both regular (noncomposite) and composite shell elements;
- provides the sensitivities of selective design responses with respect to the appropriate design variables for each of the above analysis types;
- is available for static stress, transient dynamic, frequency, and direct-solution steady-state dynamic analysis using models that have only stress/displacement elements;
- is available for workflows involving a combined Abaqus static and fe-safe fatigue analysis;
- accounts for contact, geometric, and some material nonlinearities; and
- is often carried out within a combined Abaqus-Tosca design optimization workflow but can also be used in a stand-alone Abaqus/Standard analysis to output design sensitivities.

Adjoint design sensitivity analysis is typically used in sensitivity analysis having many design parameters. Alternatively, you can use *Direct Design Sensitivity Analysis* in Abaqus/Standard for parametric design optimization having only a few design parameters and many design responses.

Adjoint design sensitivity analysis is used to compute the derivatives of design responses with respect to one or more design variables. These derivatives are commonly referred to as sensitivities, because they provide a first-order measure of how sensitive the design response is to a change in the design variables. The adjoint design sensitivity analysis capability is useful when the design optimization methodology requires a relatively small set of design responses, but a large number of design variables, typically one or more variables associated with each element or node of the design domain. This is the case for topology, sizing, shape, bead, and ABD stiffness sensitivities, where the adjoint method for computing design sensitivities is significantly more cost effective computationally compared to the direct method. More details on the different kinds of structural optimization methods can be found in *About Structural Optimization*. However, adjoint sensitivity analysis can also be used in a stand-alone Abaqus/Standard analysis (that is, outside of an iterative design optimization workflow using Abaqus/Standard and Tosca) to obtain design sensitivities at any stage during the design process.

Material Options

The adjoint sensitivity calculations are supported with elastic, hyperelastic, and isotropic hardening Mises plasticity material models. With plasticity, the sensitivity calculations use a total formulation that assumes that the loading is monotonic (that is, no unloading) and proportional. The sensitivity results are not accurate if the above assumptions are not valid. More generally, for material constitutive models with path-dependent and irreversible response, the adjoint sensitivity calculations are supported only for monotonic loading.

Output

The responses and response sensitivities are output only to the output database (`.odb`) file; sensitivity output to the data (`.dat`) file and results (`.fil`) file is not supported.

The design responses are labeled using the following format:

$$\text{DResp_\{}response_identifier\text{\} _\{}response_name\text{\}}$$

The ***response_identifier*** refers to one of the design responses listed in the tables included in *Specifying Design Responses in a Static or an Eigenfrequency Analysis* and *Specifying Design Responses in a Transient Dynamic Analysis*; for example, U1 for the 1-component of the displacement vector at a node, RF2 for the 2-component of the reaction force vector at a node. The ***response_name*** refers to the name you specified in the design response definition. The response name helps distinguish among similar design responses.

The labels of the sensitivities are related to the labels of the design responses as follows:

$$\text{d_\{}response_identifier\text{\} _\{}response_name\text{\}_\{}design_variable_name\text{\}}$$

The ***design_variable_name*** depends on the type of sensitivity analysis as follows:

- For topology sensitivity, the design variable name is prefixed by Abaqus/Standard to be `_density`. For example, if the design response is U1, the corresponding sensitivity is `d_U1_{response_name}__density`.
- For shape and bead sensitivities, the design variable name is prefixed by Abaqus/Standard to be `_position`. In this case, the sensitivity output represents a vector with three components in the *X*-, *Y*-, and *Z*-directions. For example, if the design response is U1, the corresponding shape or bead sensitivities are a vector `d_U1_{response_name}__position`.
- For sizing sensitivity with respect to either the total thickness for noncomposite shells or an individual layer thickness for composite shells, the design variable name is prefixed by Abaqus/Standard to be `_thickness`.
- For sizing sensitivity with respect to the material orientation angle in a layer of a composite shell, the design variable name is prefixed by Abaqus/Standard to be `_angle`.
- For ABD stiffness sensitivity with respect to the stiffness components of a shell general section, the design variable name is prefixed by Abaqus/Standard to be `_stiff`. In this case, the sensitivity output represents a vector with twenty-one components corresponding to the stiffness components. For example, if the design response is U1, the corresponding ABD stiffness sensitivities are a vector `d_U1_{response_name}__stiff`.

The ***response_name*** in the label for the sensitivities helps distinguish among design sensitivities that are of the same type.

Limitations

Adjoint design sensitivity analysis is subject to a number of limitations.

Sizing sensitivity with composite shells and ABD

- Cannot be used in an eigenfrequency analysis.
- Cannot be used in a direct steady-state dynamic analysis.

stiffness sensitivity for shell elements defined using shell general section:

- Cannot be used with inertia relief loads.

Loads and Constraints:

- Inertia relief loads are supported only for three-dimensional elements in linear perturbation steps and load cases.
- Nodes that are involved in a multi-point constraint definition, either as a main node or a secondary node, cannot be part of the design domain for a shape or bead sensitivity analysis.
- Topology and sizing design variables cannot be used with the same element in a linear perturbation static analysis with inertia relief loading.

Materials:

- The isotropic Mises plasticity model is the only supported plasticity material model. For this model, the sensitivity calculations are based on a total formulation that assumes that all material points undergo proportional loading with no unloading.
- For adjoint sensitivity calculations, numerical damping contributions are taken into account consistently for moderate dissipation or quasi-static applications of a transient dynamic analysis. While consistent numerical damping contributions are also included during the adjoint sensitivity calculations of transient fidelity applications, such problems often experience numerical instabilities during adjoint sensitivity calculations with thin-walled structures (for example, shells). In transient dynamic simulations for which the transient fidelity application type is the default choice, you should use a moderate dissipation application type for accurate adjoint sensitivities.

Elements:

- Continuum shell elements are not supported for sizing sensitivities.
- The following element types are not supported for shape sensitivities: continuum elements with incompatible modes; generalized plane strain elements; cylindrical solid elements; axisymmetric solid elements with nonlinear, asymmetric deformation; continuum solid shell elements; cylindrical membrane elements; and axisymmetric membrane elements (with and without twist).
- The following element types are not supported for bead sensitivities: continuum shell elements; axisymmetric shell elements; axisymmetric shell elements with nonlinear, asymmetric deformation; conventional three-dimensional shell elements with five degrees of freedom per node; and triangular facet thin shell elements (STRI3).
- The internal force design response cannot be associated with elements that have secondary nodes (for any feature that uses a main-secondary concept, such as a multi-point constraint).

Eigenfrequency analysis:

- Only the Lanczos or AMS eigensolver can be used during an eigenfrequency analysis.
- Eigenfrequency response is not available for a preloaded structure.

Transient analysis:

- In a transient dynamic procedure, shape or bead sensitivity analysis is supported only for moderate dissipation and quasi-static applications.
- A transient dynamic shape sensitivity analysis is not supported for the following element types: continuum elements with incompatible modes; generalized plane strain elements; cylindrical solid elements; and axisymmetric solid elements with nonlinear, asymmetric deformation; continuum solid shell elements; cylindrical membrane elements; and axisymmetric membrane elements (with or without twist).

- A transient dynamic bead or shape sensitivity analysis is not supported with thermal loads.
- A transient dynamic sensitivity analysis must contain only a single dynamic step.
- For a transient sensitivity analysis, the specification of the time points can contain only a single time point.
- Material damping is not supported for transient sensitivity calculations.
- For three-dimensional Timoshenko beam elements, a transient sensitivity analysis supports only isotropic rotary inertia. As a result the rotary inertia associated with the torsional mode is used for all rotational degrees of freedom.

Direct steady-state dynamic adjoint sensitivity analysis:

- Only supports nodal (translational and rotational) displacement, velocity, and acceleration responses in sizing, shape, and bead sensitivity analysis. Reaction forces and moments are not supported as design responses.
- The use of multiple domains is not supported.
- Design-dependent loads and connector elements in the model are not supported. The node sets specified in the node responses cannot contain any nodes with dependent degrees of freedom. The aggregation operators over the node sets of the node responses are limited to sum (default), maximum, and average.
- Preloading and design-dependent loads are not supported.

Use with restart analyses:

- In a restart analysis, you cannot request adjoint design response and sensitivities if the original analysis did not include such requests.
- You cannot request adjoint design response and sensitivities that are based on the combined responses of a perturbation step and a preceding general step if the perturbation step appears in a restart analysis while the preceding general step appears in the original analysis.

Static fatigue responses using fe-safe:

- Abaqus/Standard supports fatigue design responses for only a static perturbation analysis with multiple load cases.
- You can include only one fatigue design response type in an Abaqus model. However, you can include several different fatigue responses of the same type (for example, DAMAGE) but having different element sets and different design response operators in the same Abaqus model.
- Abaqus/Standard does not support fatigue design response and adjoint sensitivity calculations for composite sizing (layer thicknesses and angles) design variables.

Input File Template

Input file template for sensitivity analysis in a static procedure:

```
*HEADING
…
*NODE, NSET=nset
Data lines to define the nodes
*ELEMENT, TYPE=element type, ELSET=elset_elastic
Data lines to define the elements
*SOLID SECTION, ELSET=elset_elastic, MATERIAL=elastic
*MATERIAL, NAME=elastic
*ELASTIC
Data lines to define the elastic properties
*STEP, SENSITIVITY=ADJOINT
```

```
*STATIC
…
*DESIGN RESPONSE, NAME=response name
*ELEMENT RESPONSE, ELSET=elset name
Data line to specify the element response identifier key
*DESIGN RESPONSE, NAME=response name
*ELEMENT RESPONSE, ELSET=elset name, NSET=nset name
Data line to specify the element response identifier key
*DESIGN RESPONSE, NAME=response name
*NODE RESPONSE, NSET=nset
Data line to specify the nodal response identifier key
*DESIGN RESPONSE, NAME=response name
*GLOBAL RESPONSE
Data lines to specify the global response identifier keys
*END STEP
```

Input file template for sensitivity analysis in a transient dynamic procedure:

```
*HEADING
…
*NODE, NSET=nset
Data lines to define the nodes
*ELEMENT, TYPE=element type, ELSET=elset_elastic
Data lines to define the elements
*SOLID SECTION, ELSET=elset_elastic, MATERIAL=elastic
*MATERIAL, NAME=elastic
*ELASTIC
Data lines to define the elastic properties
*DENSITY
Data lines to define density properties
*TIME POINTS, NAME=TimePoint_Name1
Data lines to define a list of time points
*TIME POINTS, NAME=TimePoint_Name2
Data lines to define a list of time points
*STEP, SENSITIVITY=ADJOINT
*DYNAMIC
…
*DESIGN RESPONSE, NAME=response name, TIME MARKS=YES, TIME POINTS=TimePoint_Name1
*ELEMENT RESPONSE, ELSET=elset name
Data line to specify the element response identifier key
*DESIGN RESPONSE, NAME=response name, TIME MARKS=YES, TIME POINTS=TimePoint_Name2
*NODE RESPONSE, NSET=nset
Data line to specify the nodal response identifier key
*END STEP
```

Input file template for adjoint sensitivity analysis in a direct steady-state dynamic procedure:

```
*HEADING
…
*NODE, NSET=nset
Data lines to define the nodes
*ELEMENT, TYPE=element type, ELSET=elset_elastic
Data lines to define the elements
*SOLID SECTION, ELSET=elset_elastic, MATERIAL=elastic
*MATERIAL, NAME=elastic
*ELASTIC
Data lines to define the elastic properties
*DENSITY
```

```
Data lines to define density properties
*STEP, SENSITIVITY=ADJOINT
*STEADY STATE DYNAMICS, DIRECT
…
*DESIGN RESPONSE, NAME=response name
*NODE RESPONSE, NSET=nset, OPERATOR=operator label, RANGE OPERATOR=range operator label
Data line to specify the nodal response identifier key
frequency lower limit, frequency upper limit
*END STEP
```

Input file template for fatigue design sensitivity analysis:

```
*HEADING
  …
*NODE, NSET=nset
 Data lines to define the nodes
*ELEMENT, TYPE=element type, ELSET=elset_elastic
 Data lines to define the elements
*SOLID SECTION, ELSET=elset_elastic, MATERIAL=elastic
*MATERIAL, NAME=elastic
*ELASTIC
 Data lines to define the elastic properties
*STEP, PERTURBATION SENSITIVITY=ADJOINT
*STATIC
…
*LOAD CASE, NAME=name1
…
*END LOAD CASE
*LOAD CASE, NAME=name2
…
*END LOAD CASE
…
*DESIGN RESPONSE, NAME=response name
*ELEMENT RESPONSE, FATIGUE, ELSET=elset name
 Data line to specify the element response identifier key
*END STEP
```

Adjoint Sensitivity Method for Transient Dynamics Using Abaqus/Standard

The adjoint sensitivity approach for transient dynamic problems is based on the implicit dynamic procedure in Abaqus/Standard.

Some classes of problems are better suited than others for such an approach. These problems include low frequency–mode bending-dominated deformations and low-velocity crash problems (such as side crashes and other bending-dominated crashes). Other problems in this category include battery frame crashes, pedestrian head impacts, dynamics of machinery, and compliant flexible dynamic mechanisms. Typically, such problems use 30–300 increments to capture the correct physics and applying Abaqus/Standard is appropriate. Therefore, the adjoint sensitivity method for transient dynamics is applicable for these situations.

However, for high frequency–mode crash problems (for example, axial deformation and crushing), very small time increments are required to capture the correct physics. The run time for many practical applications of this latter type are not feasible for Abaqus/Standard due to a very large number of time increments. Therefore, Abaqus/Explicit is better suited to solve this class of problems. Also, Abaqus/Standard might often fail to converge when the models have significant nonlinearities (many contacts and many buckling modes). Such problems are also solved more effectively using Abaqus/Explicit.

Adjoint Sensitivity Method for Fatigue Workflows Using Abaqus and fe-safe

Computing fatigue design responses and their corresponding sensitivities requires execution of both Abaqus and fe-safe.

Abaqus computes the stresses, and fe-safe then does the fatigue calculations, based upon these stresses. You can trigger this workflow by specifying a fatigue element design response in Abaqus.

As shown in *Figure 1*, the preprocessing module in Abaqus starts the fe-safe executable. The fe-safe fatigue calculations require an fe-safe macro file and an fe-safe project archive to be present prior to the start of the analysis.

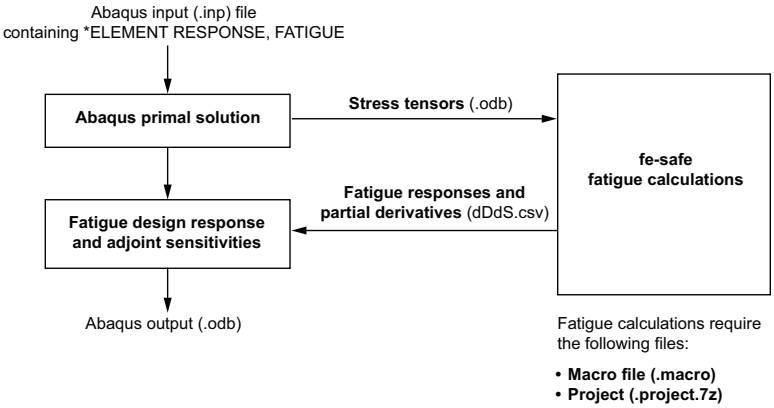


Figure 1: Fatigue adjoint sensitivity workflow.

As shown in the example below, entries in an fe-safe `.macro` file do the following:

- define the fatigue response types;
- request that fe-safe write the fatigue response and corresponding partial derivatives with rest to the stress tensors of the various load cases in the file `dDdS.csv`; and
- specify the compute configuration for the fe-safe calculations (note that the compute configuration for fe-safe is not affected by the compute configuration for Abaqus/Standard).

```
[stress perturbation.variables]="< fe-safe variable name >"
[stress perturbation.Sensitivities File] = ..\dDdS.csv
[project.job.solver limit] = 8
```

The fatigue response types in the Abaqus design response definition and in the fe-safe .macro file must be consistent. The label for the Abaqus design response and the name of the corresponding fe-safe variable in the .macro file are both case insensitive.

By default, the name of the fe-safe macro file ends with `.macro`, and the name of the fe-safe project file ends with `.project.z7`. The first part of both file names should match the name of the Abaqus input (`.inp`) file. If the fe-safe macro and project files are in the directory where the model is run, these files are picked up automatically by the combined Abaqus and fe-safe workflow. Alternatively, you can provide the fe-safe macro and project file using the command line arguments `-fes_macro` and `-fes_project`, respectively, and by providing the complete path to these files. In this case, the first parts of the file names do not need to match the input file name.

Available fatigue design responses for combined Abaqus/fe-safe workflow are discussed in *Specifying Design Responses in a Combined Abaqus Static and fe-safe Fatigue Analysis*.

Activating Adjoint Sensitivity Analysis

You activate an adjoint sensitivity analysis on a step-by-step basis.

Input File Usage: Use the following option to activate an adjoint sensitivity analysis in a particular step:

**STEP*, SENSITIVITY=ADJOINT

Activating Adjoint Sensitivity Analysis in Multiple Steps

If you activate an adjoint sensitivity calculation in any general step during the analysis, it is activated for all general steps in the analysis, including steps before the activating step. Likewise, if you activate an adjoint sensitivity calculation in any perturbation step during the analysis, it is activated for all perturbation steps in the analysis, including steps before the activating step. In both of these situations, the sensitivity calculations occur for any given step that includes responses. In a linear perturbation step with multiple load cases, the design sensitivity calculations are carried out for all the load cases. The adjoint sensitivity calculations cannot be selectively deactivated for some of the load cases.

Specifying Design Variables

You can specify design variables in an adjoint design sensitivity analysis.

Unlike direct design sensitivity analysis (which is commonly used for parametric design and optimization), adjoint design sensitivity analysis is commonly used for nonparametric design and optimization. The design parameters are more commonly referred to as design variables in this case. Unlike direct design sensitivity analysis (where the number of design parameters is relatively small in number), an adjoint design sensitivity analysis typically has one or more design variables associated with each element in the design domain.

The specific choices for design variables depend on the type of design sensitivities that you desire, as listed in the table below. For topology, sizing, and ABD stiffness sensitivities, the design variables and the corresponding design sensitivities are element-based, while for shape and bead sensitivities these quantities are node-based. A model can also include mixed design variables for topology, sizing, ABD stiffness, and shape or bead sensitivities.

Type of sensitivity	Design variables
Topology	Relative material density
Sizing for noncomposite shells	Shell element thickness
Sizing for circular section beams	Beam element radius
Shape or bead	Nodal positions
Sizing for composite shells	Composite layer thickness and/or orientation
ABD stiffness	Components of the symmetric half of the $[\mathbf{D}]$ matrix for the shell general section

Topology Sensitivity

The design variables for topology optimization are the material relative densities for the elements in the design domain. The material relative density is a nondimensional quantity that can take a value that is greater than 0 but less than or equal to 1. A value of 1 corresponds to the original material density, while a value that is close to 0 corresponds to removal of material to create a void. For computing design response and sensitivities, Abaqus/Standard makes use of scaling factors and their derivatives for stiffness, mass, and stress design response, respectively. The scaling factors and their derivatives are all functions of the relative material density, and you can use a spatial distribution to specify them directly. All elements in the design domain must explicitly specify a value for the different scaling factors and their derivatives as part of the spatial distribution definition. An element for which the scaling factors and its derivatives are not specified is assumed not to be a part of the design domain.

Input File Usage: Use the following options to define the stiffness scaling factors and their derivatives for topology sensitivities:

**SCALE STIFFNESS*
distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the mass scaling factors and their derivatives for topology sensitivities:

**SCALE MASS*
distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution*

name, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the stress design response scaling factors and their derivatives for topology sensitivities:

**SCALE STRESS DESIGN*
distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Sizing Sensitivity with Noncomposite Shell Elements

The design variables for sizing with noncomposite shell elements are the shell thicknesses. All elements in the design domain must explicitly specify a value for the shell thickness using a spatial distribution. An element for which the thickness is not specified in the spatial distribution definition is assumed not to be a part of the design domain.

Input File Usage: Use the following options to define the design variables for sizing sensitivity for shell elements with the section behavior integrated during the analysis:

**SHELL SECTION*, SHELL THICKNESS=*distribution name*
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the design variables for sizing sensitivity for shell elements with general section behavior:

**SHELL GENERAL SECTION*, SHELL THICKNESS=*distribution name*
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Sizing Sensitivity with Composite Shell Elements

The design variables for sizing with composite shell elements are the composite layer thicknesses and the orientation angles. All elements in the design domain must explicitly specify a value for the layer thickness and/or orientation angle using spatial distributions. If you do not specify the layer thickness or orientation angle in the spatial distribution definition for an element, Abaqus/Standard assumes that element is not part of the design domain.

Input File Usage: Use the following options to define the design variables for the layer sizing for composite shell elements with the section behavior integrated during the analysis:

**SHELL SECTION*, COMPOSITE
distribution name, number of integration points, name of the material forming this layer, orientation of this layer
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the design variables for the layer sizing for composite shell elements with general section behavior:

**SHELL GENERAL SECTION*, COMPOSITE
distribution name, , name of the material forming this layer, orientation of this layer
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the design variables for the layer orientation angle for composite shell elements with the section behavior integrated during the analysis:

**SHELL SECTION*, COMPOSITE
layer thickness, number of integration points, name of the material forming this layer, distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Use the following options to define the design variables for the layer orientation angle for composite shell elements with general section behavior:

**SHELL GENERAL SECTION*, COMPOSITE
layer thickness, , name of the material forming this layer, distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Sizing Sensitivity with Circular Section Beam Elements

The design variables for sizing with circular section beam elements are the beam radii. All elements in the design domain must explicitly specify a value for the beam radius using a spatial distribution. An element for which the radius is not specified in the spatial distribution definition is assumed not to be a part of the design domain. Adjoint design sensitivity calculations with circular section beams is supported only for elements with section behaviors that are integrated during the analysis.

You must request output variable BRADIUS, which contains the beam radius for a circular cross-section beam, to render (visualize) the beam elements with varying radii properly in the Visualization module of Abaqus/CAE. Otherwise, all beam radii are rendered with a value of 1.0.

Input File Usage: Use the following options to define the design variables for sizing optimization with circular section beam elements:

**BEAM SECTION*, SECTION=CIRC
distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

ABD Stiffness Sensitivity with Shell General Sections

The design variables for ABD stiffness sensitivities for shell general sections are the components of the symmetric half of the $[\mathbf{D}]$ matrix. All elements in the design domain must specify values for the section stiffness components explicitly using a spatial distribution. Elements for which the section stiffness components are not specified in the spatial distribution definition are assumed not to be a part of the design domain.

Input File Usage: Use the following options to define the design variables for ABD stiffness optimization with shell elements defined using a general shell section:

**SHELL GENERAL SECTION*
distribution name
**DISTRIBUTION*, LOCATION=ELEMENT, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Shape Sensitivity with Solid Elements

The design variables for shape sensitivity analysis with solid elements are the positions (coordinates) for all nodes that are in the design domain. All nodes in the design domain must explicitly specify, as part of a spatial distribution, a value of the displacement components relative to their original locations. A node for which the displacement is not specified through a spatial distribution definition is assumed to not be a part of the design domain.

Input File Usage: Use the following options to define the design variables for shape optimization with solid elements:

**ADJUST*, DISTRIBUTION=*distribution name*, NODE SET=*name*
**DISTRIBUTION*, LOCATION=NODE, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Bead Sensitivity with Shell Elements

The design variables for a bead sensitivity analysis with shell elements are the positions (coordinates) of all nodes that are in the design domain. All nodes in the design domain must explicitly specify, as part of a spatial distribution, a value of the displacement components relative to their original locations. A node for which the displacement is not specified through a spatial distribution definition is assumed to not be a part of the design domain.

Input File Usage: Use the following options to define the design variables for bead optimization with shell elements:

**ADJUST*, DISTRIBUTION=*distribution name*, NODE SET=*name*
**DISTRIBUTION*, LOCATION=NODE, DESIGN VARIABLE, NAME=*distribution name*, TABLE=*distribution table name*
**DISTRIBUTION TABLE*, NAME=*distribution table name*

Specifying Design Responses in a Static or an Eigenfrequency Analysis

An adjoint design sensitivity analysis supports three kinds of design responses: element responses, global responses, and node responses.

The following element responses are valid:

- von Mises stress, signed von Mises stress (the sign of this quantity is the same as the sign of the trace of the stress tensor),
- maximum and minimum principal stress,
- maximum and minimum principal strain,
- maximum and minimum principal logarithmic strain,
- maximum and minimum principal nominal strain,
- plastic strain magnitude,
- Neuber and Glinka plasticity correction quantities (*Plasticity Corrections*), and
- a user-defined response based on either the stress, the strain, the logarithmic strain, the nominal strain, or the plastic strain; and internal forces (see *Defining the Design Response in User Subroutine UELEMDRESP*)..

A stress-, strain-, logarithmic strain-, nominal strain-, or plastic strain–based design response represents a scalar measure calculated using a specified operator (by default, the maximum) on the values in the specified element set. Abaqus/Standard uses the stresses, the strains, the logarithmic strains, the nominal strains, or the plastic strains at the element integration points to compute stress-, strain-, logarithmic strain-, nominal strain-, or plastic strain–based design responses and sensitivities. The internal forces response represents a scalar measure calculated using a specified operator (by default, the sum) on the internal nodal forces over a set of specified nodes from a specified set of elements.

Only two types of global responses are available: stiffness and eigenfrequency. The global stiffness response, available in a static procedure, provides a measure of the stiffness of the structure that accounts for both external loading and prescribed displacements. It is defined as

$$G_{stiff} = \frac{1}{2}\mathbf{P} \cdot \mathbf{U} - \frac{1}{2}\mathbf{R}_F \cdot \mathbf{U}_{pres},$$

where $\mathbf{P}$, $\mathbf{U}$, $\mathbf{R}_F$, and $\mathbf{U}_{pres}$ refer to vectors representing external loads, displacements, reaction forces, and prescribed displacements on the structure. For a structure that has only external loading and no prescribed displacements, the second term is identically zero, and only the first term produces a nonzero contribution. The global eigenfrequency response, available in an eigenfrequency procedure, refers to one or more eigenfrequencies of the structure.

The node responses refer to quantities such as displacements or reaction forces and moments at one or more nodes in the structure.

You request design responses using a syntax analogous to that for specifying output requests to the output database. There are no default responses. If no responses are requested, no design responses and no sensitivities are output. Specifying a response outputs both the design response value and the response sensitivities.

For internal forces and internal moments in element responses, both an element and a node set must be specified to define the design response. Each node in the specified node set receives contributions from all the elements in the specified element set that are attached to it. For element responses and node responses, if the node set contains more than one node, the design response contains the sum of the contributions from all the nodes in the node set.

By default, the sensitivity for the stress, the strain, the logarithmic strain, the nominal strain, and the plastic strain magnitude responses are approximated with aggregation functions over the specified element set, where the default is an aggregation function representing the maximum. Other aggregation functions approximating the minimum and the difference between the maximum and minimum values are also available. You apply these operator types for the design response. Additional operators for elemental design responses are the sum and the average over the elemental integration points of a given element set.

Input File Usage: Use the following options to request element design responses:

**DESIGN RESPONSE*, NAME=*response name*
**ELEMENT RESPONSE*, ELSET=*elset name*, NSET=*nset name*,
ORIENTATION=*orientation name*, OPERATOR=*operator name*

Use the following options to request global design responses:

**DESIGN RESPONSE*, NAME=*response name*
**GLOBAL RESPONSE*

Use the following options to request node design responses:

**DESIGN RESPONSE*, NAME=*response name*
**NODE RESPONSE*, NSET=*nset name*, ORIENTATION=*orientation name*

Each design response specification block must only request a single design response. Multiple design responses require multiple uses of the design response specification block, once for each response.

Defining the Design Response in User Subroutine `UELEMDRESP`

You can specify a user-defined element response in user subroutine *UELEMDRESP*. This response can be based on stress, strain, logarithmic strain, nominal strain, or plastic strain. You specify an element response identifier label (such as `TRESCA`, `MISES_PE`, or `DRESP_1`), the name of a table collection that refers to a property or parameter table, and output variable S, E, LE, NE, or PE. Abaqus activates the specified table collection before it calls the user subroutine. If you do not specify a table collection, Abaqus activates the table collection, if any, that is associated with the material definition of the elements in the design domain.

Input File Usage: Use the following options to define the element design response in user subroutine *UELEMDRESP*:

**DESIGN RESPONSE*, NAME=*response_name*
**ELEMENT RESPONSE*, USER, ELSET=*elset_name*
element response identifier label, table collection name, output variable identifier

For example, the following input defines a Tresca stress invariant as the element design response:

**DESIGN RESPONSE*, NAME=*response_name*
**ELEMENT RESPONSE*, USER, ELSET=E1
TRESCA, TC_NAME, S

Requesting Responses in Multiple Steps

Unless specified otherwise, response requests defined in a step propagate to subsequent steps according to the following rules:

1. Requests in general steps propagate to subsequent general steps.
2. Requests in linear perturbation steps propagate to subsequent consecutive linear perturbation steps.

Responses in a Linear Static Perturbation Step

For sensitivities in a static linear perturbation step following a general static step, the design responses are computed based on the total values of the appropriate physical quantities. Here, total refers to the sum of the physical quantities from the general and the perturbation steps. This approach is particularly useful when the general step represents the preloading and the assembly process of a structure while the perturbation step, which

is typically composed of a number of load cases, represents service and live loads. The perturbation step may or may not apply the Linear Complementarity Problem (LCP) solution technique (see *Linear Complementarity Problem (LCP) Solution Technique for Solving Contact Problems*).

Restrictions on Responses

The tables below describe the available design responses for various procedure types.

Table 1: Node responses for a static procedure.

Physical quantity	Node response
Displacement components	U1, U2, U3
Displacement magnitude	UMAG
Rotation components	UR1, UR2, UR3
Rotation magnitude	URMAG
Reaction force components	RF1, RF2, RF3
Reaction force magnitude	RFMAG
Reaction moment components	RM1, RM2, RM3
Reaction moment magnitude	RMMAG

Table 2: Element responses for a static procedure.

Physical quantity	Element response
Internal force components	INTF1, INTF2, INTF3
Internal force magnitude	INTFMAG
Internal moment components	INTM1, INTM2, INTM3
Internal moment magnitude	INTMMAG
Von Mises equivalent stress	MISES
Signed von Mises equivalent stress	S_MISES
Maximum principal stress	SP3
Minimum principal stress	SP1
Maximum principal strain	EP3
Minimum principal strain	EP1
Maximum principal logarithmic strain	LEP3
Minimum principal logarithmic strain	LEP1
Maximum principal nominal strain	NEP3
Minimum principal nominal strain	NEP1
Neuber equivalent stress	NBSEQ
Glinka equivalent stress	GKSEQ
Neuber equivalent strain	NBEEQ
Glinka equivalent strain	GKEEQ
Plastic strain magnitude	PEMAG
Neuber equivalent plastic strain	NBPEEQ
Glinka equivalent plastic strain	GKPEEQ

Table 3: Global responses for a static or frequency procedure.

Procedure type	Physical quantity	Global response
Static	Global stiffness measure	STIFF
Frequency	Eigenfrequency	FREQ

Specifying Design Responses in a Combined Abaqus Static and fe-safe Fatigue Analysis

A fatigue adjoint design sensitivity analysis for linear load cases supports element responses.

The following fatigue element responses are valid:

- damage for the entire fatigue loading,
- maximum damage out of any of the blocks for the fatigue loading,
- reciprocal of radial fatigue reserve factor ($\frac{1}{FRF_R}$), and
- reciprocal of vertical fatigue reserve factor ($\frac{1}{FRF_V}$).

A fatigue design response represents a scalar measure calculated using a specified operator (by default, the maximum) on the fatigue values in the specified element set. Abaqus/Standard uses the fatigue responses and the partial derivatives with respect to the stress tensors at the element integration points (which are provided by an fe-safe analysis) to compute fatigue-based design responses and corresponding sensitivities.

You must request a fatigue design response explicitly in your simulation—otherwise Abaqus/Standard will not output any response or sensitivities related to fatigue. Specifying a fatigue design response results in the output of both the response value and the corresponding sensitivities.

Because the fatigue computations in fe-safe applies the stresses for all the load cases in the Abaqus simulationto the fatigue design response calculations, identical response values and corresponding sensitivities of the fatigue design responses are assigned to all the load cases in the output database.

By default, Abaqus/Standard approximates the sensitivities of the fatigue responses with an operator over the specified element set, where the default is the maximum operator. Other available operators include minimum, difference between maximum and minimum, the sum, and the average over the fatigue responses for all the integration points of the given element set

Input File Usage: Use the following options to request fatigue element design responses:

**DESIGN RESPONSE*, NAME=*response name*, TIME MARKS=YES, TIME POINTS=*time point name*
**ELEMENT RESPONSE*, ELSET=*elset name*, OPERATOR = *operator name*, FATIGUE
fatigue_element_response_identifier_label

Restrictions on Responses

The table below describes the available fatigue design responses for load cases of a linear static procedure.

Table 1: ELement responses and corresponding fe-safe variable names for a static perturbation procedure with multiple load cases.

Physical quantity	Element response	fe-safe variable name
Damage for the entire fatigue loading	DAMAGE	Damage
Maximum damage out of any of the blocks for the fatigue loading	MAXBLOCKDAMAGE	MaxBlockDamage
Reciprocal of radial fatigue reserve factor	FRFRADINV	1/FRF-Radial
Reciprocal of vertical fatigue reserve factor	FRFVERTINV	1/FRF-Vert

Specifying Design Responses in a Transient Dynamic Analysis

A transient dynamic adjoint design sensitivity analysis supports two kinds of design responses: node responses and element responses. The node responses refer to quantities such as displacements or reaction forces and moments at one or more nodes in the structure.

You request design responses using a syntax analogous to that for specifying output requests to the output database. There are no default responses. If no responses are requested, no design responses and no sensitivities are output. Specifying a response outputs both the design response value and its sensitivities. You must specify a node set or an element set as part of a node or an element design response definition. If the set contains more than one entity (node or element), the design response contains the sum of the contributions from all of the entities in the set.

Optionally, you can specify one or more time points at which a physical quantity of interest is to be treated as a design response. If no time point is specified, the time point corresponding to the end of the step is used for the design response.

Input File Usage:

Use the following options to request element design responses at a specific time point:

**DESIGN RESPONSE*, NAME=*response name*, TIME MARKS=YES, TIME POINTS=*time point name*
**ELEMENT RESPONSE*, ELSET=*elset name*

Use the following options to request node design responses at a specific time point:

**DESIGN RESPONSE*, NAME=*response name*, TIME MARKS=YES, TIME POINTS=*time point name*
**NODE RESPONSE*, NSET=*nset name*, ORIENTATION=*orientation name*

Each design response specification block must request only a single design response. Multiple design responses require multiple uses of the design response specification block, once for each response.

Defining the Design Response in User Subroutine `UELEMDRESP`

You might also use user subroutine *UELEMDRESP* to include user-defined element design response, based on stress, strain, logarithmic strain, nominal strain, or plastic strain (see *Defining the Design Response in User Subroutine UELEMDRESP*).

Restrictions on Responses

The tables below describe the available design responses for transient dynamic analysis.

Table 1: Node design responses for transient dynamic analysis.

Physical quantity	Node response
Displacement components	U1, U2, U3
Displacement magnitude	UMAG
Rotation components	UR1, UR2, UR3
Rotation magnitude	URMAG
Translational velocity components	V1, V2, V3
Translational velocity magnitude	VMAG
Rotational velocity components	VR1, VR2, VR3
Rotational velocity magnitude	VRMAG

Physical quantity	Node response
Translational acceleration components	A1, A2, A3
Translational acceleration magnitude	AMAG
Rotational acceleration components	AR1, AR2, AR3
Rotational acceleration magnitude	ARMAG
Reaction force components	RF1, RF2, RF3
Reaction moment components	RM1, RM2, RM3

Table 2: Element design responses for transient dynamic analysis.

Physical quantity	Element response
Rate of internal work	ELPF
Von Mises equivalent stress	MISES
Signed von Mises equivalent stress	S_MISES
Maximum principal stress	SP3
Minimum principal stress	SP1
Maximum principal strain	EP3
Minimum principal strain	EP1
Maximum principal logarithmic strain	LEP3
Minimum principal logarithmic strain	LEP1
Maximum principal nominal strain	NEP3
Minimum principal nominal strain	NEP1
Neuber equivalent stress	NBSEQ
Glinka equivalent stress	GKSEQ
Neuber equivalent strain	NBEEQ
Glinka equivalent strain	GKEEQ
Plastic strain magnitude	PEMAG
Neuber equivalent plastic strain	NBPEEQ
Glinka equivalent plastic strain	GKPEEQ

Specifying Design Responses in a Direct Steady-State Dynamic Analysis

A direct steady-state dynamic adjoint design sensitivity analysis supports only node responses. The node responses refer to quantities such as displacements, velocities, or accelerations at one or more nodes in the structure.

For each design response, you specify a range of frequencies over which the sensitivities are computed. The output values of a design response and the corresponding sensitivities are aggregates (default is maximum) computed over the range of frequencies specified for the design response.

There are no default responses. If no responses are requested, no design responses and no sensitivities are output. Specifying a response outputs both the design response value and its sensitivities. You must specify a node set for each node response requested. If the node set has multiple nodes, the design response contains the aggregate (default is sum) of the contributions from all of the nodes in the node set.

The frequency range, which you specify with a node response request, must include one or more frequency values at which the primal steady-state dynamic response is computed. If this is not the case, Abaqus/Standard outputs the design response and sensitivities as zeros. You might also obtain the primal steady-state dynamic response at frequency values that are outside of the range specified with the node response definition; these have no effect on the design response and sensitivity output.

The node set you specify for any node response should not contain any node with a dependent degree of freedom. There should be no preloading in an analysis with a direct steady-state dynamic adjoint design sensitivity analysis step. Connector elements and design dependent loads (such as gravity loading, GRAV) are not supported in a direct steady-state dynamic adjoint design sensitivity analysis. You can run a direct steady-state dynamic analysis on multiple domains; however, if you request an adjoint sensitivity analysis for this procedure, you can run the analysis only on a single domain. The permitted aggregation operators over the node set are sum (SUM), maximum (MAX), and average (AVE).

Input File Usage: Use the following options to request a node design response over a frequency range:

**DESIGN RESPONSE*, NAME=*response name*
**NODE RESPONSE*, NSET=*nset name*, OPERATOR=*operator label*, RANGE OPERATOR=*range operator label*
node response label
frequency range lower limit, *frequency range higher limit*

Each design response specification block must request only a single design response. Multiple design responses require multiple uses of the design response specification block, once for each response.

Restrictions on Responses

The table below describes the available design responses for a direct steady-state dynamic procedure.

Table 1: Node responses for a direct steady-state dynamic procedure.

Physical quantity	Node response
Displacement components	U1, U2, U3
Displacement magnitude	UMAG
Rotation components	UR1, UR2, UR3
Rotation magnitude	URMAG
Velocity components	V1, V2, V3
Velocity magnitude	VMAG

Physical quantity	Node response
Rotational velocity components	VR1, VR2, VR3
Rotational velocity magnitude	VRMAG
Acceleration components	A1, A2, A3
Acceleration magnitude	AMAG
Rotational acceleration components	AR1, AR2, AR3
Rotational acceleration magnitude	ARMAG

Limiting Sensitivity Output by Specifying Bounds on Design Response Values

By default, Abaqus writes the design response and the corresponding adjoint sensitivities to the output database. In some situations, the sensitivities are necessary only if the value of the design response is within a certain range. You can specify such a range through the specification of a lower bound and an upper bound with the response definition.

The bounds provide a convenient way to limit the output in the output database in situations where the sensitivity values are not necessary. The following provides some additional details related to the specification of bounds for design response values:

- If no bounds are specified, then the adjoint sensitivities are always calculated and written to the output database.
- If the design response value is lower than the "lower bound," then only the design response value is written to the output database.
- If the design response value is higher than the "upper bound," then only the design response value is written to the output database.
- If both the lower bound and the upper bound are specified to have a value of zero, then only the design response value is written to the output database.

The specification of lower and upper bounds is supported for node, element, and global design responses in a static and transient sensitivity analysis. It is not supported for modal eigenfrequency and a steady-state dynamic sensitivity analysis.

For a multiple load case analysis, a design response is defined outside of the individual load case definitions. In such situations, the design response values and corresponding sensitivities are typically output for all load cases. If bounds are specified for a design response, the adjoint sensitivities for this specific design response are written for all the load cases if the response value for any one of the load cases satisfies the bound.

Input File Usage: Use the following options to specify lower or upper bounds on node response:

**NODE RESPONSE*, LOWER BOUND=*lower_bound_value*, UPPER BOUND=*upper_bound_value*

Use the following options to specify lower or upper bounds on element response:

**ELEMENT RESPONSE*, LOWER BOUND=*lower_bound_value*, UPPER BOUND=*upper_bound_value*

Use the following options to specify lower or upper bounds on global response:

**GLOBAL RESPONSE*, LOWER BOUND=*lower_bound_value*, UPPER BOUND=*upper_bound_value*

Parametric Studies

In this section:

- *Scripting Parametric Studies*
- *Parametric Studies: Commands*

Scripting Parametric Studies

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Parametric Modeling*
- *Parametric Shape Variation*
- *Parametric Studies*

Overview

Parametric studies allow you to generate, execute, and gather the results of multiple analyses that differ only in the values of some of the parameters used in place of input quantities.

Parametric studies can be performed by:

- Creating a "template" parametrized input file from which the different parametric variations are generated.
- Preparing a script (a file with the `.psf` extension) that contains Python (*Lutz, 1996*) instructions to generate, execute, and gather output for the parametric variations of the parametrized input file.

The Python commands for scripting parametric studies are discussed in this section.

Introduction

Parametric studies require that multiple analyses be performed to provide information about the behavior of a structure or component at different design points in a design space. The inputs for these analyses differ only in the values assigned to the parameters of a parametrized keyword input file (identified with the `.inp` extension).

Parametric studies in Abaqus require a user-developed Python script in a file (identified with the `.psf` extension) that contains Python commands to define the parametric study. For example, consider a case where you wish to perform a parametric study in which the thickness of a shell is varied. You need to create a parametrized input file (in this example, a file named `shell.inp`) containing the parameter definition

```
*PARAMETER
thick1 = 5.
```

and the parameter usage:

```
*SHELL SECTION,ELSET=name, MATERIAL=name
<thick1>
```

You create the parametric study by developing a `.psf` file that contains a script of Python instructions specifying the different designs that are to be analyzed, as follows:

```
thick = ParStudy(par='thick1', name='shell')
thick.define(CONTINUOUS, par='thick1', domain=(10., 20.))
thick.sample(NUMBER, par='thick1', number=5)
thick.combine(MESH)
```

These scripting commands create five designs with corresponding section thicknesses of 10., 12.5, 15., 17.5, and 20.0. Each of these thicknesses will, in turn, replace the value of 5. specified in the parameter definition in `shell.inp`. You may then provide additional Python scripting commands in the `.psf` file instructing Abaqus to do the following:

- Generate a number of `shell_`*id*`.inp` files and corresponding Abaqus jobs using the `shell.inp` file as a template. (The identifier *id* that is appended to the input file name is unique to each design in the parametric study.) An example of the Python command for this is

  ```
  thick.generate(template='shell')
  ```

  In this example the `shell_`*id*`.inp` files will differ only in the value to be used for the shell thickness.
- Execute all the Abaqus jobs representing the different variations of the parametric study. The Python command for this is

  ```
  thick.execute(ALL)
  ```

You generally want to review certain key results from the large amount of data that is generated by a parametric study. Abaqus provides the following capabilities for this purpose:

- A command specifying the source from which the results of a parametric study will be gathered. For example:

  ```
  thick.output(file=ODB, step=1, inc=LAST)
  ```

  The command above sets the output location to the last frame of the first step in the output database (`.odb`) file. The default behavior is to gather results from the last frame of a given step in the results (`.fil`) file.
- Commands to gather the required results from the multiple analyses generated by the parametric study and report them in a file or table. For example, the sequence of Python scripting commands used to gather and report the value of a displacement at a key node for each of the designs is:

  ```
  thick.gather(results='n33_u', variable='U', node=33, step=1)
  thick.report(PRINT, par='thick1', results=('n33_u.2'))
  ```

  The commands above gather the results record `'n33_u'` (the displacement vector of node 33 at the end of Step 1 of the analysis) for each of the designs and then print a table of the U2 component (the second component of the results record) of displacement for all designs.
- The ability to visualize *X*–*Y* plot data gathered across multiple analyses using the Visualization module of Abaqus/CAE. A typical example is to obtain an *X*–*Y* plot of the value of the displacement at a key node versus the value of the shell thickness. This is done by gathering the appropriate parametric study results in an ASCII file that can be read into the Visualization module to display the plot.

Organization of Parametric Studies

A parametric study in Abaqus is associated with a particular set of parameters that define the design space. Only the values of the parameters can change in a parametric study. A new parametric study must be created if you wish to consider a different set of parameters. Having selected the parameters to be considered in a parametric study, you must specify how each parameter is defined. Parameters are distinguished as either continuous or discrete in nature and may have a domain and reference value.

The design points in the design space that are to be analyzed are created by specifying sample values for each parameter (sampling) and by combining the parameter samples to create sets of design points. A few simple commands are provided for parameter value sampling and for combining the sampled parameter values; these commands are described in detail later.

An initial definition and sampling of the parameters in the parametric study must be given before any combinations of parameter samples can be specified. After the first combination the initial definition and/or sampling of any individual parameter can be changed before the next combination is specified, thus providing a great deal of flexibility within one parametric study.

The domain of possible values and the reference value for a parameter given in the parameter definition can be temporarily redefined in any sampling of that parameter by specifying them differently during the sampling.

You need not specify the parameter domain and reference value in the parameter definition so long as these are specified during sampling.

Design constraints can be imposed on all of the designs. A design that violates any of the constraints will be eliminated.

Finally, after all parametric study variations have been analyzed, you can gather and report results across all or some of the designs of the parametric study.

In summary, parametric studies in Abaqus are organized as follows:

- Create parametric study.
- Define parameters: define parameter type (continuous or discrete valued) and possibly the parameter domain and reference value.
- Sample parameters: specify sampling option and data and possibly temporarily redefine the parameter domain and reference value.
- Combine parameter samples to create sets of designs.
- Constrain designs (optional).
- Generate designs and analysis job data.
- Execute the analysis jobs for selected designs of the study.
- Gather key results for selected designs of the study.
- Report gathered results.



Note:

The sequence of steps—define, sample, and combine—can be repeated as often as is necessary to create all the required design sets. Multiple parametric studies can be performed on a model contained in one input file. In general, more parameters will be defined and used in place of input quantities in the input file than those involved in any particular parametric study. In these cases parameters not involved in a particular parametric study will retain their values defined in the input file for the purposes of that parametric study. Therefore, we can think of the parameter values defined in the input file as representing a nominal design; parametric studies create modified designs by overwriting the values of some (or all) parameters.

Defining the Design Space

The design space is defined by the selection of the parameters to be varied in the study as well as the specification of the parameter types and possible values they can have.

Parametric Study Creation

Use the *aStudy*=`ParStudy` scripting command (see *aStudy=ParStudy()*) to create a parametric study and select the independent parameters to be considered for variation. *aStudy* is the Python variable name assigned by you to the parametric study object created by the command. The methods of the parametric study object are used to carry out all the actions of the parametric study.

Input File Usage: *aStudy*=`ParStudy` (**par=**, **name=**, **verbose=**, **directory=**)

Parameter Definition

Use the *aStudy*`.define` command (see *aStudy.define()*) to specify the parameter type (choose the CONTINUOUS or the DISCRETE token; a token is a symbolic constant used to select an option within a specific command)

and, optionally, to specify the domain of possible parameter values and a reference value for the parameter. If the domain and/or reference value are not specified in this command, they can be specified in the parameter sampling.

Redefinitions of a parameter are treated as complete redefinitions; that is, no information is retained from the previous definition of that parameter.

Input File Usage: *aStudy*`.define` (*token*, **par=**, **domain=**, **reference=**)

CONTINUOUS Parameter Type

In this case the parameter can take any value in a continuous domain specified by minimum and maximum values; for example, **domain=**(3., 10.).

DISCRETE Parameter Type

In this case the parameter can take only the values specified in a list that defines the discrete domain; for example, **domain=**(1, 4, 9, 16).

Sampling and Combining Parameter Values to Create Sets of Design Points

Each parameter in the parametric study must be sampled before the combination operation is used to create the first set of design points. Any parameter in the parametric study can be redefined or resampled before a subsequent combination operation is performed.

Parameter Sampling

Use the *aStudy*`.sample` command (see *aStudy.sample()*) and choose one of the available tokens (INTERVAL, NUMBER, REFERENCE, or VALUES) to select how the sampling is done. The sampling data that must be given depend on how the sampling is done, as described next.

Sampling by INTERVAL

This sampling command assumes that you specify a domain of possible parameter values and wish to sample parameter values at fixed intervals in the domain. Sampling of the extreme values of the parameter is always done. The number of parameter values sampled depends on the interval and the domain. Because the extreme values are sampled, the last sampling interval will generally be smaller than the interval you specify.

The domain specification in this sampling command is optional:

- If a domain is specified in this command, it temporarily redefines a domain specified in the `define` command.
- If a domain is not specified in this command, the domain specification from the `define` command is used for sampling.
- An error is flagged when a domain is not specified in this command or in the `define` command.

The sampling interval is interpreted differently for continuous and discrete parameters:

- For continuously valued parameters the interval at which the samples are spaced is based on a numerical value. For example, specifying **interval=**10. for a continuous parameter with **domain=**(10., 35.) will sample values of 10., 20., 30., and 35. for this parameter.
- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values. The index means the position of the entry in the list, starting at position 0 and continuing with positions 1, 2, 3, etc. In this case **interval** must be an integer number. For example, specifying **interval=**–2

for a discrete parameter with **domain**=(1., 2., 3., 5., 7., 10.) will create sample values of 10., 5., 2., and 1. for this parameter.

The interval can have a positive or negative value (zero is not permitted). A positive interval indicates that sampling starts at the minimum value for a continuous parameter or at the first value in the list of values for a discrete parameter (forward sampling). A negative interval indicates that sampling starts at the maximum value for a continuous parameter or at the last value in the list of values for a discrete parameter (reverse sampling). Reverse sampling is useful when the TUPLE combination operation is used (see the discussion of combination of parameter samples).

Two special cases of the INTERVAL option are noteworthy:

- A positive interval value larger than the range of continuous parameter values or the number of discrete parameter values will sample the minimum and maximum values of the continuous parameter or the first and last values in the discrete parameter list.
- A negative interval value larger (in absolute terms) than the range of continuous parameter values or the number of discrete parameter values will sample the maximum and minimum values of the continuous parameter or the last and first values in the discrete parameter list.

Input File Usage: *aStudy*`.sample` (INTERVAL, **par**=, **interval**=, **domain**=)

Sampling by NUMBER

This sampling option assumes that you specify a domain of possible parameter values and wish to sample a fixed number of parameter values in the domain. Except for a special case documented below, sampling of the extreme values of the parameter is always done. The parameter is sampled at equally spaced intervals (with some exceptions for discrete parameters, as discussed below) and the size of the interval depends on the number of values sampled as well as the domain.

The domain specification in this sampling command is optional:

- If a domain is specified in this command, it temporarily redefines the domain specified in the `define` command.
- If a domain is not specified in this command, the domain specification from the `define` command is used for sampling.
- An error is flagged when a domain is not specified in this command or in the `define` command.

The sampling interval is calculated and interpreted differently for continuous and discrete parameters:

- For continuous valued parameters the interval at which the samples are spaced is based on a numerical value. For example, specifying **number**=4 for a continuous parameter with **domain**=(10., 25.) will sample values of 10., 15., 20., and 25. for this parameter.
- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values (indexing starts at zero). For example, specifying **number**=3 for a discrete parameter with **domain**=(1., 2., 3., 5., 7., 10., 12.) will create sample values of 1., 5., and 12.for this parameter. The number of discrete parameter samples specified by you may not allow equally spaced sampling; for example, specifying **number**=5 or **number**=6 for the discrete parameter above does not allow equally spaced sampling. This is resolved by sampling the parameter values that are closest to being equally spaced by rounding the sampling index to the closest index in the list of values. For example, specifying **number**=5 for the discrete parameter above will create sample values of 1., 3., 5., 10., and 12. The values 1. and 12. are sampled because they are the extreme values. The explanation for the second sampled value being the third value in the list (the value 3.) is as follows: the sampling interval is (highest index − lowest index)/(**number** − 1) = (6 − 0)/(5 − 1) = 1.5; the second sampled value should then be the one with index = 0 + 1.5 = 1.5 in the list; since the index has to be an integer number, we round off to index = 2 and, thus, sample the third value in the list. The other

sampled values can be explained similarly. The same rule is used for character string type discrete parameters. For example, specifying **number**=3 for a discrete parameter with **domain**=('C3D8', 'C3D8R', 'C3D8I', 'C3D8H') will create sample values of 'C3D8', 'C3D8I', and 'C3D8H'.

Three special cases of the NUMBER option are noteworthy:

- Specifying **number**=1 will sample the central value of a parameter, which is useful when the center of the design space is of interest. It is the only case in which the use of the NUMBER option does not sample the extreme values of the parameter.
- Specifying **number**=2 will sample the extreme values of a parameter, which is useful when the boundaries of the design space are of interest.
- Specifying **number**=3 will sample the central and the extreme values of a parameter, which is useful when the center and the boundaries of the design space are of interest.

Specification of **number**=0 is not permitted. A negative value for **number** is permitted; this indicates that the sampling is to be in reverse order. For continuous parameters reverse order means that the first sampled value is the largest and the last sampled value is the smallest. For discrete parameters reverse order means that the first sampled value is the last in the list of values and the last sampled value is the first in the list of values. Sampling in reverse order is useful when the TUPLE combination operation is used (see the discussion of combination of parameter samples).

Input File Usage: *aStudy*`.sample` (NUMBER, **par**=, **number**=, **domain**=)

Sampling by REFERENCE

This sampling option allows you to specify a reference value for the parameter and to sample parameter values with respect to this reference value. It is useful for studying alternate designs with respect to an existing (reference) design.

This sampling command creates sample values symmetrically about the reference value at multiples of a given interval; in addition, the reference value is also sampled. The number of parameter values in the sample depends on the number of symmetrical pairs of values you specify.

The reference value specification in this sampling option is optional:

- If a reference value is specified in this command, it temporarily redefines the reference specified in the `define` command.
- If a reference value is not specified in this command, the reference specification from the `define` command is used for sampling.
- An error is flagged when a reference value is not specified in this command or in the `define` command.

The reference value is interpreted differently for continuous and discrete parameters:

- For continuous valued parameters **reference** is the parameter's numerical value about which a symmetrical sample will be created.
- For discrete valued parameters **reference** is the index of the list of values about which a symmetrical sample will be created.

A reference value that falls outside the domain definition for the parameter is flagged as an error.

The sampling interval is interpreted differently for continuous and discrete parameters:

- For continuous valued parameters the interval at which the samples are taken is based on a numerical value. For example, specifying **reference**=50., **interval**=10., and **numSymPairs**=2 for a continuous parameter will create sample values of 30., 40., 50., 60., and 70. for this parameter.

- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values (indexing starts at zero); in this case **interval** must be an integer value. For example, specifying **reference**=5, **interval**=−2 and **numSymPairs**=2 for a discrete parameter with **domain**=[1, 2, 3, 5, 7, 10, 12, 15, 20, 25] will create sample values of 25, 15, 10, 5, and 2 for this parameter.

The specified **interval** can have a positive or negative value, but a value of zero is not permitted. A positive interval indicates that the list of sampled values starts with the smallest sampled value for a continuous parameter or with the sampled value closest to the beginning of the list of values for a discrete parameter (forward sampling). A negative interval indicates that the list of sampled values starts with the largest sampled value for a continuous parameter or with the sampled value closest to the end of the list of values for a discrete parameter (reverse sampling). Reverse sampling is useful when the TUPLE combination operation is used (see the discussion on combination of parameter samples).

The number of symmetrical pairs you specify must be zero or a positive integer; setting the number of symmetrical pairs equal to zero indicates that only the reference value is sampled.

The domain specification in this command is optional:

- If a domain is specified in this command, it temporarily redefines the domain specified in the `define` command.
- If a domain is not specified in this command, the domain specification from the `define` command is used for sampling.
- An error is flagged in the case of discrete valued parameters when a domain is not specified in this command or in the `define` command.

A domain specification (either in this command or in the `define` command) is required for discrete valued parameters because the possible discrete values that can be sampled must be known. Although a domain specification is not required for continuous valued parameters, it may be given. In either the case of discrete parameters or the case of continuous parameters, a domain specification can be used to limit the number of values sampled using the REFERENCE option since the domain is treated as a bound on the possible sampling values. For example, specifying **reference**=50., **interval**=10., and **numSymPairs**=3 for a continuous parameter with **domain**=(35., 100.) will sample values of 40., 50., 60., 70., and 80. for this parameter. The minimum value of the domain acts as a bound in this sampling.

Input File Usage: *aStudy*`.sample` (REFERENCE, **par**=, **reference**=, **interval**=, **numSymPairs**=, **domain**=)

Sampling by VALUES

This sampling option assumes that you wish to create the parameter sample values directly. You must specify the actual parameter values, irrespective of whether the parameter is continuous or discrete.

A parameter domain specified in the `define` command does not affect the values sampled for the parameter when this option is used.

Input File Usage: *aStudy*`.sample` (VALUES, **par**=, **values**=)

Combination of Parameter Samples

Use the *aStudy*`.combine` command (see *aStudy.combine()*) to create sets of design points from the parameter samples. Choose how the combining is done using one of the following tokens: MESH, TUPLE, or CROSS. The use of each combination command results in the creation of a number of design points, which are grouped into design sets. If a combine operation creates a design that duplicates a design in an existing design set, the

duplicate design is deleted immediately. The total number of designs in a parametric study (before the application of any design constraints) is the sum of the number of designs in each design set.

You can name a design set; if you do not, it is named by default. The default naming convention is *p1* for the first non-user-named design set in the parametric study, *p2* for the second non-user-named design set, and so on. The design set name is used to help identify individual designs. A design set named by you with a name identical to a previously specified design set name indicates that it is a respecification of the design set and, thus, overwrites the previously existing one.

Input File Usage: *aStudy*`.combine` (*token*, **name**=)

MESH Combination

This combine option indicates that every sampled value for a parameter is to be combined with every sampled value of every other parameter in the parametric study.

The following examples illustrate the use of the MESH combine option. In a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13))
study.sample(REFERENCE, par='par1', reference=0,
    interval=2, numSymPairs=2)
study.define(CONTINUOUS, par='par2', domain=(10., 60.))
study.sample(INTERVAL, par='par2', interval=20.)
```

the combine command

```
study.combine(MESH, name='dSet1')
```

creates the following 12 design points (`par1`, `par2`): (1, 10.), (5, 10.), (9, 10.), (1, 30.), (5, 30.), (9, 30.), (1, 50.), (5, 50.), (9, 50.), (1, 60.), (5, 60.), and (9, 60.) (see *Figure 1*).

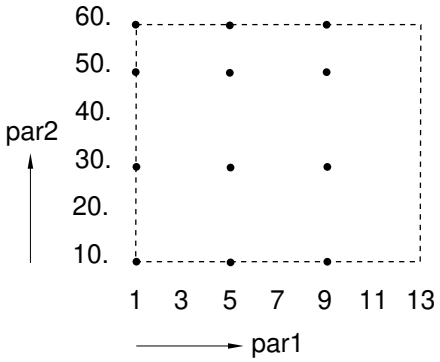


Figure 1: Design points in design set `dSet1` created with the MESH option of the combine command.

A second use of the combine command preceded by a respecification of the parameter sampling

```
study.sample(NUMBER, par='par1', number=3)
study.sample(NUMBER, par='par2', number=3)
study.combine(MESH, name='dSet2')
```

creates designs at the following nine points: (1, 10.), (7, 10.), (13, 10.), (1, 35.), (7, 35.), (13, 35.), (1, 60.), (7, 60.), and (13, 60.) (see *Figure 2*). The extreme and center values of both parameters are combined.

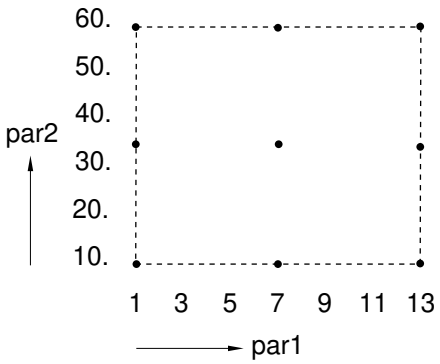


Figure 2: Design points in design set `dSet2` created with the MESH option of the combine command after the parameter sampling is redefined.

TUPLE Combination

This combine option creates design sets consisting of *n*-tuples of the sampled parameter values, where *n* is the number of parameters in the parametric study. Each *n*-tuple consists of one sampled value for each parameter. For example, in a three-parameter study the first sampled value of each of the three parameters makes up the first 3-tuple, the second sampled value of each of the three parameters makes up the second 3-tuple, and so on. The creation of tuples ceases when any of the parameter samples runs out of sampled values.

The following examples illustrate the use of the TUPLE combination operation. For a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13))
study.define(CONTINUOUS, par='par2', domain=(10., 60.))
study.sample(INTERVAL, par='par1', interval=1)
study.sample(INTERVAL, par='par2', interval=10.)
```

the combination operation

```
study.combine(TUPLE, name='dSet3')
```

creates designs at the following 6 points: (1, 10.), (3, 20.), (5, 30.), (7, 40.), (9, 50.), and (11, 60.) (see *Figure 3*). This represents a diagonal pattern in the two-parameter space. We see that all `par2` values are used in the tuple combination but the last `par1` value is not used because there are no more `par2` sample values to form additional tuples.

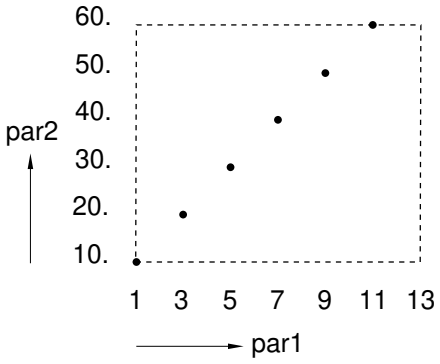


Figure 3: Design points in design set `dSet3` created with the TUPLE option of the combine command.

A second invocation of the above combine command after respecifying the `par2` sampling as

```
study.sample(INTERVAL, par='par2', interval=-10.)
study.combine(TUPLE, name='dSet4')
```

creates designs at the following 6 points: (1, 60.), (3, 50.), (5, 40.), (7, 30.), (9, 20.), and (11, 10.) (see *Figure 4*). This represents the other diagonal in the two-parameter space.

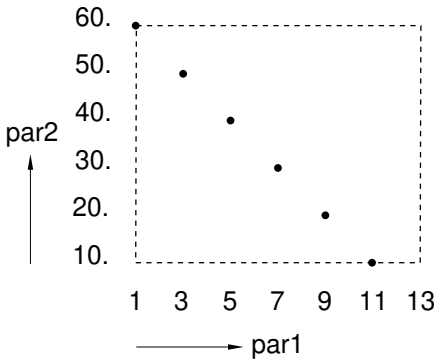


Figure 4: Design points in design set `dSet4` created with the TUPLE option of the combine command after the parameter sampling is redefined.

CROSS Combination

This combine option creates designs in the form of “cross-shaped” patterns as follows: each value sampled for an individual parameter is combined with the reference value, as specified in the `define` command, of all the other parameters used in the parametric study. To use the CROSS combine option, it is necessary to specify a reference value in the `define` command for all parameters in the parametric study.

The reference value specified for a parameter in the `define` command does not have to coincide with a value sampled by that parameter's sampling rule. However, if the reference value does not coincide with a sampled value, the reference parameter value is not added to the list of sampled values for that parameter; it is used only to form the CROSS combination.

The following examples illustrate the use of the CROSS combine option. For a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13), reference=3)
study.define(CONTINUOUS, par='par2',
    domain=(10., 60.), reference=40.)
study.sample(REFERENCE, par='par1', interval=1,
    numSymPairs=3)
study.sample(INTERVAL, par='par2', interval=10.)
```

the combine cross option

```
study.combine(CROSS, name='dSet5')
```

creates designs at the following 12 points: (1, 40.), (3, 40.), (5, 40.), (7, 40.), (9, 40.), (11, 40.), (13, 40.), (7, 10.), (7, 20.), (7, 30.), (7, 50.), and (7, 60.) (see *Figure 5*). This combination is a cross-shaped pattern where the cross intersection is at (7, 40.) as specified [7 is the fourth value (**reference**=3) of the discrete parameter `par1`].

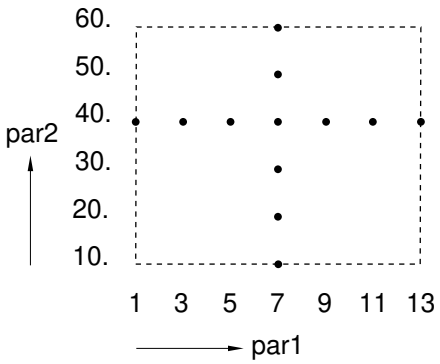


Figure 5: Design points in design set `dSet5` created with the CROSS option of the combine command.

A second invocation of the above combine command after respecifying `par2` as

```
study.define(CONTINUOUS, par='par2', domain=(10., 60.),
    reference=45.)
study.combine(CROSS, name='dSet6')
```

creates designs at the following 13 design points: (1, 45.), (3, 45.), (5, 45.), (7, 45.), (9, 45.), (11, 45.), (13, 45.), (7, 10.), (7, 20.), (7, 30.), (7, 40.), (7, 50.), and (7, 60.) (see *Figure 6*).

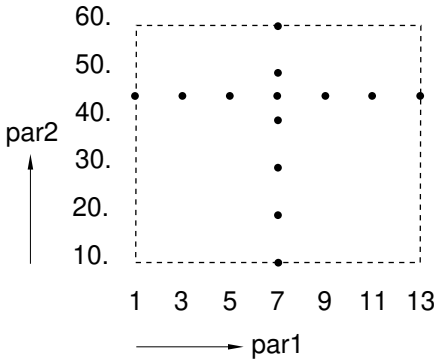


Figure 6: Design points in design set `dSet6` created with the CROSS option of the combine command when the parameter `par2` is redefined.

Constraining Designs

A constraint that determines the allowable design points in the parametric study can be specified using the *aStudy*`.constrain` scripting command (see *aStudy.constrain()*). When such a constraint is specified, existing designs that violate the constraint are eliminated immediately.

For example, the constrain command

```
aStudy.constrain('height*width < 12.')
```

where `height` and `width` are parameters in the parametric study, can be used to enforce that the cross-sectional area of a rectangular beam must be less than 12.0 in all designs.

Input File Usage: *aStudy*`.constrain` ('*constraint expression*')

Generation and Execution of the Designs of a Parametric Study

Once the required design points have been specified, it is necessary to generate the corresponding analysis job data and execute the analyses.

Generation of Analysis Job Data

Use the *aStudy*`.generate` scripting command (see *aStudy.generate()*) to generate an input file for each design.

The name of the parametrized template input file from which the input files of each design are generated must be specified.

The naming convention for the input files generated by the parametric study is as follows:

- The name of each analysis job will start with the template input file name; for example, `shell`.
- The name of the parametric study (specified by you in the `ParStudy` command using the **name=** option) is appended, preceded by an underscore (_); for example, `shell_thickness` for a parametric study defined with the `study = ParStudy(name='thickness')` command. (If no parametric study name has been given, the parametric study name defaults to the name of the Python script file in which the study is defined.) If the template input file name and parametric study name are the same, the name is not repeated.
- The name of the design set (specified or created by default in the `combine` command) is appended, preceded by an underscore (_); for example, `shell_thickness_p1` for the first design set (named by default) of the above parametric study.
- The design name (created automatically in the `combine` command) is appended, preceded by an underscore (_); for example, `shell_thickness_p1_c1` for the first design in the first design set of the above parametric study.

As usual, the input files each have the extension `.inp`. You can examine and/or edit these input files before execution.

The `generate` command creates a file with the `.var` extension that contains a description of the parametric study. This file is given the parametric study name—for example, *studyName*`.var`—and contains a list of all the designs generated, together with the parameter values associated with each design. You can examine and/or edit this file; however, editing this file will affect the gathering of results across the designs of the parametric study (see “Gathering results”).

Each time the `generate` command is used, new versions of the *studyName*`.var` files are created reflecting the designs specified by all previous `combine` commands.

The define, sample, and combine steps are performed before the generate command is executed. It is, thus, possible for you to refer to parameters that do not exist or are not independent parameters in the template input file. These errors are detected and flagged by the `generate` command.

Input File Usage: *aStudy*`.generate` (**template**)

Execution of the Analysis of the Designs of a Parametric Study

Use the *aStudy*`.execute` command (see *aStudy.execute()*) to execute the analysis of the designs of the study.

The command will submit a number of analysis jobs for execution under the control of a Python process. All designs can be evaluated without further user interaction by specifying the ALL or DISTRIBUTED options of this command, or you can control the execution of the analyses interactively by specifying the INTERACTIVE option. In the interactive case you are prompted for further execution instructions. The prompt allows you to:

- Specify a number of analyses to be executed before the process pauses and prompts you again.

- Execute the remaining analyses without further user interaction.
- Specify a number of analyses whose execution is to be skipped before the process pauses and prompts you again.
- Stop execution.

The interactive option is useful because it provides the opportunity to:

- Study the results of the analyses already executed.
- Delete unnecessary files generated by the analyses to conserve disk space when many designs are being analyzed.
- Analyze only certain designs of the parametric study.

The ALL and INTERACTIVE tokens are used for sequential execution of the Abaqus analyses on your machine. The DISTRIBUTED token can be used to schedule analysis jobs on multiple machines or multiple CPUs of one machine.

The DISTRIBUTED option is available only on variants of the Linux operating system. In particular the implementation depends on the operating system for support of the `rsh`, `rcp`, and `xhost`Linux commands. Because of the use of these commands, it is necessary that for distributed execution of parametric studies the parametric study must itself be executed on your local computer. If binary results are output during the analyses, both the local and remote computers must be binary compatible. Before the distributed execution capability can be used, it is necessary to configure the appropriate queue interfaces in the Abaqus environment file.

Each analysis of a design of the parametric study that is executed when you issue the `execute` command is, by default, executed by Abaqus in background mode, irrespective of the command option used. Files created by the Abaqus analysis of each design will overwrite any existing files of the same name without you being prompted.

You can add any necessary Abaqus execution options (refer to *Abaqus/Standard and Abaqus/Explicit Execution*) to the execution command for each of the analyses by specifying them with the **execOptions** option.

Input File Usage: *aStudy*`.execute` (*token*, **files=** , **queues=** , **execOptions=**)

Parametric Study Results

Once the analyses associated with a parametric study have been executed, the variation of key results across the different designs can be examined. First, results must be gathered from the results file or output database of each of the analyses; then, these results must be reported.

The *aStudy*`.output` command (see *aStudy.output()*) can be used to specify the source of the results to be gathered. All arguments to the *aStudy*`.output` command are optional: the specification of the file, the analysis step, and the increment (for non-frequency steps) or the mode (for frequency steps) from which the results are to be gathered. If the file is not specified, the results (`.fil`) file will be used. If the step is not specified, it must be specified in the `gather` command (see the discussion that follows). The defaults of the increment (for non-frequency steps) or the mode (for frequency steps) are the last increment of the step and the first mode calculated in the step unless specified in the `gather` command. Some arguments are applicable only to the output (`.odb`) database: the instance name, the request type (field or history), the frame value where the results are to be gathered, and whether the memory used to access an output database should be overlayed when a different output database is accessed.

The specification of the source of the gathered results remains in effect for all subsequent `gather` commands until the source is respecified. Respecification of the gathering source is treated as a complete respecification; that is, nothing is retained from the previous specification of the gathering source.

Input File Usage: *aStudy*`.output` (**file=**, **instance=**, **overlay=**, **request=**, **step=**, **frameValue=** | **inc=** | **mode=**)

Gathering Results

Use the *aStudy*`.gather` command (see *aStudy.gather()*) to gather results from the results file or output database of each of the analyses.

In each use of the `gather` command you must specify a name that is associated with the gathered result record. This label is used to refer to the results record in the `report` commands.

When gathering results from the results (`.fil`) file, each result record to be gathered must be chosen by specifying one of the available output variable identifier keys appearing under the `.fil` column heading in *Abaqus/Standard Output Variable Identifiers* or *Abaqus/Explicit Output Variable Identifiers*. For example, the U or S variable identifier keys can be specified, but the U1 or S11 variable identifier keys cannot be specified. In addition, the MODAL variable identifier key can be specified to gather eigenvalue results records (those written to the results file with the record key 1980); in this case, no variable location data are required.

When gathering results from the output (`.odb`) database, each result record to be gathered is chosen by specifying one of the available output variable identifier keys appearing under the `.odb` column heading in *Abaqus/Standard Output Variable Identifiers* or *Abaqus/Explicit Output Variable Identifiers*. For field output the component must not be specified, while for history output the component number is required; for example, the U or S variable identifier keys can be specified for field output, while the U1 or S11 variable identifier keys can be specified for history output. Unless the output is at the assembly level, an instance name must be provided as an argument to the `gather` command. An exception to this instance-name requirement is the case where the output (`.odb`) database is generated from a model not defined as an assembly of part instances, which is inferred from the presence in the output database of a single assembly named Assembly-1 and a single part instance named PART–1–1. In this case you need not explicitly refer to the instance PART–1–1.

The names of components of the result records are created automatically. For example, the command

```
myStudy.gather(results='e52_stress', variable='S', element=52)
```

creates a result record `e52_stress` whose six components (in the case of a three-dimensional solid element) are named: `e52_stress.1` (the S11 stress component), `e52_stress.2` (the S22 stress component), `e52_stress.3` (the S33 stress component), `e52_stress.4` (the S12 stress component), `e52_stress.5` (the S13 stress component), and `e52_stress.6` (the S23 stress component).

The variable location data that must be given depend on the output variable identifier key specified. (Refer to *aStudy.gather()* for a description of the location data.) Enough variable location data must be given to define a unique result record.

Input File Usage: *aStudy*`.gather` (**request=**, **results=**, **step=**, **frameValue=** | **inc=** | **mode=**, **variable=**, *additional location data*)

Reporting Results

Use the *aStudy*`.report` scripting command (see *aStudy.report()*) to report results gathered from the results files of the parametric study. Use the PRINT, FILE, or XYPLOT options to specify the kind of report to be produced:

- PRINT indicates that a table of results (with headings) is to be printed to the default output device (the screen). You may wish to limit the number of columns in a table so as to make the table readable.
- FILE indicates that a table of results (with headings) is to be written to an ASCII file.
- XYPLOT indicates that a table of results (without headings) is to be written to an ASCII file that can later be read into the Visualization module in Abaqus/CAE to display *X*–*Y* plots.

Each row in a table represents a design in the parametric study. A column in a table can represent values of a parameter, values of a gathered result, or design names.

One or more parameters can be specified in the `report` command. If no parameters are specified, the default is that all parameters in the parametric study are included in the table. The column corresponding to each parameter shows the value of that parameter in each of the designs included in the table.

A design set name can be specified to restrict the rows in the table to designs that are part of the set (refer to the `combine` command described earlier). If a design set is not specified, the default is that all designs are included in the table.

Use **variations=**ON to specify that the first column in a table must show the design names. If **variations=**ON is not specified or **variations=**OFF is included, the column of design names is not included in the table.

The names of the results to be reported must be specified as a sequence; for example, the von Mises stress of element 33, the S22 stress of element 52, and the U3 displacement of node 10 are gathered in the following three separate commands:

```
myStudy.gather(results='e33_sinv', variable='SINV', element=33)
myStudy.gather(results='e52_s', variable='S', element=52)
myStudy.gather(results='n10_u', variable='U', node=10)
```

These results can be printed in a single table using the following `report` command:

```
myStudy.report(PRINT,
    results=('e33_sinv.1', 'e52_s.2', 'n10_u.3'))
```

This example shows not only how gathered results of different types (element, nodal, etc.) can be collected in a single table but also how to refer to components of results records (the von Mises stress is the first component of SINV, S22 is the second component of S, and U3 is the third component of U—refer to *Results File* or the *Abaqus Scripting User's Guide* for a description of how the results are stored in the results file and the output database, respectively).

When either the FILE token or the XYPLOT token is used, a file name must be given to specify the file to which the results are to be written. A subsequent `report` command issued in the same session using the same file name will append the new results to the file. However, a subsequent `report` command issued in a different session using the same file name overwrites the existing file.

Input File Usage: *aStudy*`.report` (*token*, **results=**, **par=**, **designSet=**, **variations=**, **file=**)

Execution of Parametric Studies

To carry out a parametric study, you must prepare a parametrized input file (*inputFile*`.inp`). This input file is the template used for the generation of the parametric variations of the study and must contain the parameter definitions necessary to use parameters in place of input quantities. The parameters must be defined in the template file; they cannot be defined in any include files that are referred to by the template file.

In addition, you must prepare a Python scripting file, *scriptFile*`.psf`, containing instructions that script the actions of the parametric study.

Typically, you prepare the Python scripting file using an editor and then invoke execution of this file using the Abaqus execution command **abaqusscript=***scriptFile*. This command starts the Python interpreter and executes the instructions in the scripting file. Alternatively, you simply start the Python interpreter, without giving a scripting file, with the Abaqus execution command **abaqusscript**. In this case the Python interpreter remains active, and you can execute additional commands interactively or execute additional commands contained in a file (for example, *fileName*) using the Python command `execfile('`*fileName*`')`. The Python interpreter can be terminated using **[Ctrl] + d** on a Linux machine or **[Ctrl] + z** on a Windows machine.

It is normally preferable to execute a previously prepared scripting file because it is likely that you will want to develop the script iteratively; in this case you simply have to go back and edit the scripting file and re-execute it until satisfied with the result.

You can monitor the progress of the parametric variation analyses using normal operating system commands.

Execution in More than One Session

You may want to gather and report results multiple times, after the parametric variations of the study have been executed. It is possible to define, generate, and execute a parametric study in one session and gather and report results in a separate session. Only the command used to create the parametric study needs to be reissued when you start a new session.

Visualization of Parametric Study Results

The results of the analysis of a particular parametric study variation can be visualized like any other results of a single analysis.

Visualization of results gathered across designs of a parametric study requires gathering of results. For visualization the results must be reported in ASCII files (using the XYPLOT option in the `gather` command), which can be read by the Visualization module in Abaqus/CAE to produce *X*–*Y* plots of results versus parameter values or design names.

Scripting Commands

Parametric studies are scripted in files with the `.psf` extension using the Python language (*Lutz, 1996*). A parametric study object, constructed from the `ParStudy` class, is provided whose methods make scripting of parametric studies straightforward; these methods are described in this chapter.

Syntax of Scripting Commands

Scripting commands generally have the following form:

```
aStudy.method (token, data)
```

aStudy is the parametric study object to which the method applies; this object is constructed using the parametric study constructor command. *method* is the method to be used; for example, `define`, `sample`, or `execute`.

Most (but not all) commands have a *token* that selects an option of the command; for example, *aStudy*`.define (CONTINUOUS, par= )` indicates that one or more continuous parameters are being defined in a parametric study. Tokens are always given in capital letters and are mutually exclusive.

For most (but not all) commands additional *data* must be specified.

Python Language Rules

The parametric study scripts contained in scripting files must follow the syntax and semantics of the Python language. Some important aspects of this language are described here (more general Python language rules are discussed in *Parametric Modeling*).

Comments

Comments must be preceded by the # symbol. The comment is understood to continue to the end of the line. For example,

```
#
# This parametric study ...
```

```
#
studyTempEffects.generate(template='shell') #use shell input file
```

Case Sensitivity

All variable and method names, tokens, and character string literals are case sensitive across all operating systems. For example,

```
study.execute( ) # is valid
study.Execute( ) # is not valid because of the capital E
```

```
study.sample(NUMBER, ...) # uses the valid token NUMBER
study.sample(number, ...) # lower case token is not valid
```

```
study.generate(template='aFile') # 'aFile' is different
study.generate(template='afile') # from 'afile'
```

Character Strings

Character strings are indicated by using paired single (' ') or double (" ") quotation marks. Backward single quotation marks (` `) cannot be used for this purpose. For example,

```
"double quoted string"
'single quoted string'
```

Printing

The Python `print` command can be used to obtain a printed representation of any Python object. For example,

```
print 'MY TEXT'
```

will print `MY TEXT` on the standard output device.

Lists and Tuples

The scripting methods for parametric studies accept integer, real, and character string types. These primitive types can, in many cases, optionally be contained within tuple or list structures. Although there are some differences between lists and tuples in Python, they can be used interchangeably in the parametric study scripting commands; they simply represent ordered sequences of items. Items in lists or tuples must be separated by commas and enclosed in parentheses or brackets. For example,

```
aStudy.define(CONTINUOUS, par=('xCoord',))# tuple contains a
                                          # single string item
aStudy.define(CONTINUOUS, par=['xCoord']) # list contains a
                                          # single string item
aStudy.define(CONTINUOUS, par=('xCoord', 'yCoord')) # tuple
aStudy.define(CONTINUOUS, par=['xCoord', 'yCoord']) # list
```

Indentation

Python uses indentation to group blocks of statements. Therefore, a Python statement should begin in the same column as the preceding statement except where grouping of statements is required by Python.

Accessing the Data of a Parametric Study

In some cases it is desirable to have direct programmatic access to the data of a parametric study. Therefore, all of the important data of the study are stored in repositories that can be accessed as data members of the parametric study object. The repositories have a similar interface and similar behavior to that of Python dictionaries.

Repository data are stored as key, value pairs; and methods are provided for accessing the repository keys and values. The syntax `aValue = aRepository[aKey]` is used to retrieve the value associated with a repository key. A list of the keys of the repository is obtained using the `keys()` method of the repository; for example, `allKeys = aRepository.keys()`. Similarly, a list of the values of the repository is obtained using the `values()` method of the repository; for example, `allValues = aRepository.values()`. The following parametric study script shows an example of how the parameter repository of a parametric study can be accessed and how a list of the parameter names and a list of the sample values for the parameter `t1` can be obtained:

```
studyTempEffect = ParStudy(par=('t1', 't2'))
studyTempEffect.define(CONTINUOUS, par=('t1', 't2'))
studyTempEffect.sample(VALUES, par='t1', values=(200.,300.,400.))
studyTempEffect.sample(VALUES, par='t2', values=(250.,350.,450.))
parRepository = studyTempEffect.parameter
listOfParameters = parRepository.keys()
t1Sample = parRepository['t1'].sample
```

The script results in the following assignments: `listOfParameters = ['t1', 't2']` and `t1Sample = [200.0, 300.0, 400.0]`. The Python `print` command can be used to obtain information on the contents of a repository.

Parametric Study Repositories

A parametric study has the following repositories and objects as data members:

- *aStudy*`.parameter`: A repository for parameter objects keyed by parameter name. Each parameter object has a `name`, `type`, `domain`, `reference`, and `sample` data member.
- *aStudy*`.designSet`: A repository for design sets keyed by design set name. Each design set is represented as a list of design points.
- *aStudy*`.job`: A repository for analysis job objects keyed by the name of the corresponding analysis input file name (without the `.inp` extension). Each job object has a `design`, `status`, `root`, `designSet`, and `designName` data member.
- *aStudy*`.resultData`: A repository for result records keyed by a name constructed by successively appending the result name, the underscore character (_), and the design name. For results retrieved from the result (`.fil`) file, each result record is in the format of the result (`.fil`) file record for the corresponding output variable. For field results retrieved from the output (`.odb`) database, each result record will be a tuple containing the components of the results. The result record for history results from the output (`.odb`) database, which can only be retrieved for a single component, will be a tuple containing a single value.
- *aStudy*`.table`: A table object containing a representation of the table formatted by the last use of the `report` command. The table object has `title`, `variation`, `designs`, and `results` data members.

References

- Lutz, M., Programming Python, O'Reilly & Associates, Inc., 1996.

Parametric Studies: Commands

In this section:

- *aStudy.combine()*
- *aStudy.constrain()*
- *aStudy.define()*
- *aStudy.execute()*
- *aStudy.gather()*
- *aStudy.generate()*
- *aStudy.output()*
- *aStudy=ParStudy()*
- *aStudy.report()*
- *aStudy.sample()*

aStudy.combine()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.combine (token, additional data)
```

Tokens

CROSS
Use this token to create designs in a "cross-shaped" pattern from the sampled values of the parameters in the parametric study.

MESH
Use this token to create designs in a "mesh" pattern in which every sampled value of a parameter is combined with every sampled value of every other parameter in the parametric study.

PRINT
Use this token to print the design points created for the parametric study.

TUPLE
Use this token to create designs in a "tuple" pattern consisting of tuples of sampled values of the parameters in the parametric study.

References:

- *Scripting Parametric Studies*

Additional Data for CROSS, MESH, and TUPLE:

Optional Data

name
Set **name** equal to the name of the design set being defined; this name must be enclosed by quotation marks. A default design set name is created if a name is not specified.

Additional Data for PRINT:

Optional Data

name
Set **name** equal to the name of the design set for which information is being printed; this name must be enclosed by quotation marks. If a design set name is not specified, information is printed for all the design sets in the parametric study.

aStudy.constrain()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.constrain (constraint expression)
```

Required Data

constraint expression
Provide a constraint expression enclosed by matching quotation marks. This expression may involve operations among parameters, numbers, and previously defined Python variables; for example, `'height*width < maxArea-2.0'`. The constraint can be an equality or an inequality.

References:

- *Scripting Parametric Studies*

aStudy.define()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.define (token, additional data)
```

Tokens

CONTINUOUS Use this token to indicate that the parameter is continuous valued.

DISCRETE Use this token to indicate that the parameter is discrete valued.

PRINT Use this token to print parameter definitions.

References:

- *Scripting Parametric Studies*

Additional Data for CONTINUOUS:

Required Data

par Set **par** equal to the name of the parameter or the sequence of parameters being defined. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a list of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

Optional Data

domain Set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, `(10., 20.)` or `[10., 20.]`.

If **domain** is omitted from this command and the parameter is later sampled using a method that requires a domain definition, the domain must be specified in the `sample` command.

reference Set **reference** equal to the reference value of the parameter.

If **reference** is omitted from this command and the parameter is later sampled using a method that requires a reference definition, the reference must be specified in the `sample` command.

Additional Data for DISCRETE:

Required Data

par Set **par** equal to the name of the parameter or the list of parameters being defined. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a list of

parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

Optional Data

domain

Set **domain** equal to the sequence of values that the parameter may have. The values must be separated by commas and enclosed by parentheses or brackets; for example, `(1., 2., 5., 3.)` or `[1., 2., 5., 3.]`.

If **domain** is omitted from this command and the parameter is later sampled using a method that requires a domain definition, the domain must be specified in the `sample` command.

reference

Set **reference** equal to the index in the sequence of parameter values. Indexing starts at zero, so that the first value of the sequence corresponds to index zero and the last value of the sequence corresponds to an index equal to the number of values in the sequence minus one.

If **reference** is omitted from this command and the parameter is later sampled using a method that requires a reference definition, the reference must be specified in the `sample` command.

Additional Data for PRINT:

Optional Data

par

Set **par** equal to the name of the parameter or the sequence of parameters whose definition is to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

If **par** is omitted, parameter definitions are printed for all parameters in the parametric study.

aStudy.**execute()**

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.execute (token, execOptions= , additional data)
```

Tokens

ALL

Use this token to sequentially execute the analyses of all the designs of the parametric study. This option is the default.

DISTRIBUTED

Use this token to execute the analyses of all designs using the specified queue interfaces of the local and/or remote computers. A similar number of analyses will be distributed to each of the specified queues.

INTERACTIVE

Use this token to sequentially execute the analyses of all the designs of the parametric study in interactive mode. In this case the process pauses to prompt you for further execution instructions. The prompt allows you to specify the number of analyses to be executed, to execute the remaining analyses, to specify the number of analyses whose execution is to be skipped, or to skip all the remaining analyses.

Optional Data

execOptions

Set **execOptions** equal to a character string of Abaqus execution options (refer to *Abaqus/Standard and Abaqus/Explicit Execution*) that are to be added to the Abaqus execution command when executing the analyses of the designs of the parametric study; this string must be enclosed in matching quotation marks.

References:

- *Scripting Parametric Studies*

Additional Data for DISTRIBUTED:

Required Data

queues

Set **queues** equal to the queue interface name or a sequence of queue interface names. If a single name is given, it must be enclosed in matching quotation marks. If a sequence of names is given, it must be enclosed in parentheses or brackets and contain queue interface names enclosed in matching quotation marks and separated by commas.

Optional Data

files

Set **files** equal to the symbolic constant or a sequence of symbolic constants that identifies the file or files that must be returned to the local computer after remote execution. The sequence items must be separated by commas, and the sequence must be enclosed in parentheses or brackets.

The allowed symbolic constants are: DAT, LOG, FIL, SEL, MSG, STA, ODB, IPM, RES, ABQ, and PAC. The default value is **files** = (DAT, FIL, LOG, ODB, SEL).

Defining Queues and Queue Interfaces:

Before being used for a distributed parametric study, queue interfaces must be defined within the **design_startup** portion of the Abaqus environment file. For example, to define a queue interface for an existing queue short on the remote computer server, the following entry in the environment file is required:

```
def onDesignStartup():
    from par_Session import Queue
    import os

    # convenience assignment
    SCRATCH = '/scratch/' + os.environ['USER']

    # create remote queue interface
    Queue(name='short_interface', hostName='server',
          driver='abaqus', queueName='short', directory=SCRATCH)
```

If, in addition, a local queue is required, the entry must be expanded to:

```
def onDesignStartup():
    from par_Session import Queue
    import os

    # convenience assignment
    SCRATCH = '/scratch/' + os.environ['USER']

    # create remote queue interface
    Queue(name='short_interface', hostName='server',
          driver='abaqus', queueName='short', directory=SCRATCH)

    # create local queue interface
    Queue(name='local_interface', driver='abaqus',
          queueName='local')

queue_name="local"
local="echo "./%S 1>%L 2>&1" | batch"
```

aStudy.gather()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.gather (request= , results= , step= , frameValue= | inc= | mode= ,
variable= , additional data)
```

Required Data

results

Set **results** equal to a name that will be used to identify the results record gathered by this command. This name must be enclosed in matching quotation marks.

variable

Set **variable** equal to an output variable identifier key; this key must be enclosed in matching quotation marks.

For gathering results from the results (`.fil`) file only those output variable identifier keys appearing under the **.fil** column heading in *Abaqus/Standard Output Variable Identifiers* or *Abaqus/Explicit Output Variable Identifiers* are available. For example, the U or S variable identifier keys can be specified, but the U1 or S11 variable identifier keys cannot be specified. In addition, the MODAL variable identifier key can be specified to gather frequency results (those written to the results file with the record key 1980); in this case no additional data are required in this command.

When gathering results from the output database (`.odb`) file, each result record to be gathered is chosen by specifying one of the available output variable identifier keys appearing under the **.odb** column heading in *Abaqus/Standard Output Variable Identifiers* or *Abaqus/Explicit Output Variable Identifiers*. For field output the component must not be specified, while for history output the component number is required; for example, the U or S variable identifier keys can be specified for field output, while the U1 or S11 variable identifier keys can be specified for history output.

Optional Data

request

This option is applicable only if the results are to be gathered from the output database file.

Set **request** equal to FIELD or HISTORY to specify whether the results must be gathered from the field data or the history data in the output database file.

If **request** is omitted from this command, the results will be gathered from the field data.

step

Set **step** equal to the analysis step number from which the results are to be gathered.

If **step** is specified in this command as well as in the `output` command, the **step** specification in this command is used.

If **step** is omitted from this command, it must have been specified in the `output` command.

Optional and Mutually Exclusive Data:

frameValue

This option is applicable only if the results are to be gathered from the output database file.

Set **frameValue** equal to the step time or frequency value of the analysis increment in the analysis step specified from which the results are to be gathered. **frameValue** can also be set equal to the

symbolic constant LAST to specify that results are to be gathered from the last increment of the step. If no results are available at the **frameValue** specified, a warning will be issued and the results will be gathered from the closest increment.

If **frameValue** is specified in this command as well as in the `output` command, the **frameValue** specification in this command is used for gathering.

If **frameValue** is omitted from this command, the results are gathered for the **frameValue** specified in the `output` command or are gathered from the last increment in the step.

inc

Set **inc** equal to the number of the analysis increment of the non-frequency analysis step specified from which the results are to be gathered across the parametric study variations. **inc** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step.

If **inc** is specified in this command as well as in the `output` command, the **inc** specification in this command is used for gathering.

If **inc** is omitted from this command, the results are gathered from the increment specified in the `output` command or are gathered from the last increment in the step.

This option is not valid for gathering history results from the output database file.

mode

Set **mode** equal to the number of the mode of the frequency analysis step specified from which the results are to be gathered across the parametric study variations.

If **mode** is specified in this command as well as in the `output` command, the **mode** specification in this command is used.

If **mode** is omitted from this command, the results are gathered from the mode specified in the `output` command or are gathered from the first mode in the step.

References:

- *Scripting Parametric Studies*

Additional Data for Element Integration Point Variables:

Required Data

element

Set **element** equal to the number of the element for which results are to be gathered.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Optional Data

centroid

Set **centroid** equal to the symbolic constant ON to indicate that the results are to be gathered at the centroid of the element. This option is valid only when the results have been written to the results file at the centroid of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

int

Set **int** equal to the number of the integration point of the element for which results are to be gathered. This option is valid only when the results have been written to the results file at the integration points of the element.

centroid, **int**, and **node** are mutually exclusive.

If **int** is omitted, the default is **int**=1. If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

node

Set **node** equal to the number of the node in the element for which results are to be gathered. This option is valid only when the results have been written to the results file at the nodes of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

rbnum

Set **rbnum** equal to the number of the rebar for which results are to be gathered. The rebar number is consistent with the order, per element, in which you define the rebar (see *Defining Rebar as an Element Property*).

If **rbnum** is omitted, the default is **rbnum**=1.

Rebar information cannot be gathered from the output database file.

rebar

Set **rebar** equal to the name of the rebar for which results are to be gathered (defined as described in *Defining Rebar as an Element Property*). Rebar results can be obtained for continuum and beam elements only at integration points; for shell and membrane elements rebar results can be obtained at integration points and at the centroid of the element.

Rebar information cannot be gathered from the output database file.

section

Set **section** equal to the number of the section point of the element for which results are to be gathered. This applies to beam, shell, or layered solid elements. **section** is not relevant for rebar results.

If **section** is omitted, the default is **section**=1.

Additional Data for Element Section Variables:

Required Data

element

Set **element** equal to the number of the element for which results are to be gathered.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Optional Data

centroid

Set **centroid** equal to the symbolic token ON to indicate that the results are to be gathered at the centroid of the element. This option is valid only when the results have been written to the results file or output database at the centroid of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

int

Set **int** equal to the number of the integration point of the element for which results are to be gathered. This option is valid only when the results have been written to the results file or output database at the integration points of the element.

centroid, **int**, and **node** are mutually exclusive.

If **int** is omitted, the default is **int**=1. If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

node

Set **node** equal to the number of the node in the element for which results are to be gathered. This option is valid only when the results have been written to the results file or output database at the nodes of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

Additional Data for Whole Element Variables:

Required Data

element

Set **element** equal to the number of the element for which results are to be gathered.

int

Set **int** = –1.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Additional Data for Partial Model (Element Set) or Whole Model Variables:

Optional Data

elset

Set **elset** equal to the element set name for which results are to be gathered. If **elset** is omitted, results will be gathered for the whole model. This name must be enclosed in matching quotation marks.

instance

This option is required only if results are gathered from an output database file and if the element set is defined on an instance.

Set **instance** equal to the name of the instance on which the element set is defined. This name must be enclosed in matching quotation marks.

If the element set is defined on the assembly, **instance** must not be specified in both this command and the `output` command.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Additional Data for Nodal Variables:

Required Data

node

Set **node** equal to the number of the node for which results are to be gathered.

instance

This option is required only if results are gathered for a node on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Additional Data for Modal Variables:

There are no additional data.

Additional Data for Contact Surface Variables:

Required Data

main

Set **main** equal to the name of the main surface of the contact pair for which results are to be gathered. This name must be enclosed in matching quotation marks.

secondary

Set **secondary** equal to the name of the secondary surface of the contact pair for which results are to be gathered. This name must be enclosed in matching quotation marks.

Optional Data

mainInstance

This option is required only if results are gathered from an output database file and if the main surface is defined on an instance.

Set **mainInstance** equal to the name of the instance on which the main surface is defined. This name must be enclosed in matching quotation marks.

secondaryInstance

This option is required only if results are gathered from an output database file and if the secondary surface is defined on an instance.

Set **secondaryInstance** equal to the name of the instance on which the secondary surface is defined. This name must be enclosed in matching quotation marks.

Required Data When Secondary Surface Node Variable Results Are Requested:

node

If secondary surface node variable results are to be gathered, set **node** equal to the number of the node for which results are to be gathered.

nset and **node** are mutually exclusive.

instance

This option is required only if results are gathered for a node on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Optional Data When Whole Surface Variable Results Are Requested:

nset

Set **nset** equal to the name of the node set for which whole surface variable results are to be gathered. This name must be enclosed in matching quotation marks.

If **nset** is omitted, the default is the whole surface.

If the results are collected from the output database file and the node set is defined on an instance, the node set name must be prefixed with the instance name and a period (for example: “PART–1–1.TOP”).

nset and **node** are mutually exclusive.

instance

This option is required only if results are gathered from an output database file and if the node set is defined on an instance.

Set **instance** equal to the name of the instance on which the node set is defined. This name must be enclosed in matching quotation marks.

If the node set is defined on the assembly, **instance** must not be specified in both this command and the `output` command.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Additional Data for Cavity Radiation Surface Variables:

Required Data

element

Set **element** equal to the number of the element underlying the cavity facet for which results are to be gathered.

elface

Set **elface** equal to the face identifier of the face of the element underlying the cavity facet for which results are to be gathered.

instance

This option is required only if results are gathered from an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the `output` command, the **instance** specification in this command is used.

Additional Data for Section File Output:

Required Data

sectionName

Set **sectionName** equal to the name of the section for which results are to be gathered. This name must be enclosed in matching quotation marks (see *Section Output from Abaqus/Standard*).

aStudy.generate()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.generate (template= )
```

Required Data

template

Set **template** equal to the name of the template input file from which the input files of each of the parametric study variations are to be generated; this name must be enclosed in matching quotation marks.

References:

- *Scripting Parametric Studies*

aStudy.output()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.output (file=, instance=, overlay=, request=, step= , frameValue= |
inc= | mode= )
```

Optional Data

file

Set **file** equal to FIL or ODB to specify that results must be read from the results (`.fil`) file or output database (`.odb`) file.

If **file** is omitted from this command, the results will be read from the results (`.fil`) file.

instance

This option is applicable only if results are gathered from an output (`.odb`) database containing multiple instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If results are gathered for a set defined on the assembly, **instance** must not be specified in both this command and the `gather` command.

If **instance** is omitted from this command and an instance name is required to gather the results, it must be specified in the `gather` command.

overlay

This option is applicable only if results are gathered from an output (`.odb`) database.

This option is used to control the trade-off between memory usage and execution speed. Set **overlay** equal to OFF (the default) or ON to specify whether the memory used for an output database should be overwritten. If memory usage is a problem, set **overlay** equal to ON to specify that the memory used for an output database must be overwritten after a `gather` is performed for the specific output database. If execution speed is more important than memory usage, set **overlay** equal to OFF to specify that memory must be allocated for each output database.

The **overlay** option affects your ability to separately manipulate the output database (`.odb`) file. When you set **overlay** equal to OFF, the file will remain open after accessing the results from the `gather` command, which may affect your ability to separately access or manipulate the file (to delete the file, for example). Set **overlay** equal to ON to separately access the `.odb` file after accessing the `gather` based results.

request

This option is applicable only if the results are to be gathered from the output (`.odb`) database.

Set **request** equal to FIELD or HISTORY to specify whether the results must be gathered from the field data or the history data in the output database.

If **request** is omitted from this command, the results will be gathered from the field data.

step

Set **step** equal to the analysis step number from which the results are to be gathered.

If **step** is omitted from this command, it must be specified in the `gather` command.

Optional and Mutually Exclusive Data:

frameValue This option is applicable only if the results are to be gathered from the output (`.odb`) database.

Set **frameValue** equal to the step time or frequency value of the analysis increment in the analysis step specified from which the results are to be gathered. **frameValue** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step. If no results are available at the **frameValue** specified, a warning will be issued and the results will be gathered from the closest increment.

If **frameValue** is omitted from this command, the results are gathered from the last increment in the step or are gathered from the increment specified in the `gather` command.

inc

Set **inc** equal to the number of the analysis increment of the non-frequency analysis step specified from which the results are to be gathered. **inc** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step specified.

If **inc** is omitted from this command, the results are gathered from the last increment in the step or are gathered from the increment specified in the `gather` command.

This option is not valid for gathering history results from the output (`.odb`) database.

mode

Set **mode** equal to the mode number of the frequency analysis step specified from which the results are to be gathered across the parametric study variations.

If **mode** is omitted, the results are gathered from the mode specified in the `gather` command or are gathered from the first mode in the step.

References:

- *Scripting Parametric Studies*

aStudy=ParStudy()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy=ParStudy (par= , name= , verbose= , directory= )
```

Required Data

par

Set **par** equal to the sequence of independent input parameters selected for the parametric study. This sequence must be given inside parentheses or brackets and must contain independent parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`. If only one parameter is to be listed, its name can be given enclosed by matching quotation marks; for example, `'par1'`.

Optional Data

name

Set **name** equal to the name of the parametric study; the name must be enclosed in matching quotation marks. If a name is not specified, its value defaults to the name of the Python script file that contains the parametric study commands.

verbose

Set **verbose** equal to the symbolic token OFF to suppress the printing of comment and warning messages. The default is **verbose**=ON.

directory

Set **directory** equal to the symbolic token ON to select that subdirectories of the current directory are used to organize the files of the parametric study. A subdirectory will be created for each design that is analyzed. The default is **directory**=OFF.

References:

- *Scripting Parametric Studies*

aStudy.**report()**

Products: Abaqus/Standard Abaqus/Explicit

Command

aStudy.report (*token*, **results**= , **par**= , **designSet**= , **variations**=, **truncation**= , *additional data*)

Tokens

FILE

Use this token to specify that results are to be written to an ASCII file as a table with the relevant headings.

PRINT

Use this token to specify that results are to be printed as a table with the relevant headings. Since the results are printed to the default output device (the screen), you may wish to limit the number of columns in a table so as to make the table readable.

XYPLOT

Use this token to specify that results are to be written to an ASCII file as a table without headings. This table can subsequently be read by the Visualization module in Abaqus/CAE to display *X–Y* plots of result and parameter values.

Required data

results

Set **results** equal to the sequence of result names to be reported; this sequence must be enclosed by parentheses or brackets. For example, `results=('e33_sinv.1', 'e52_strain', 'n25_u.3')`, where `'e33_sinv.1'` is the von Mises stress of element 33 (Mises is the first component of the SINV record), `'e52_strain'` are all the strain components of element 52, and `'n25_u.3'` is the third component of displacement of node 25. This example assumes that the three results above were gathered in previous `gather` commands by requesting the SINV, E, and U variable identifier keys, respectively.

Optional data

par

Set **par** equal to the name of the parameter or the sequence of parameters to be included in the report table. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

If **par** is omitted, all parameters in the parametric study are included in the report table.

designSet

Set **designSet** equal to the name of the design set whose results are to be included in the report table; this name must be enclosed by matching quotation marks.

If **designSet** is omitted, results for all design sets in the parametric study are included in the report table.

variations

Set **variations** equal to ON to indicate that the first column of the report table must show the names of the designs being reported. Set **variations** equal to OFF to indicate that the names of the designs being reported are not to be given in the first column of the report table.

If **variations** is omitted, the column of design names is not included in the report table.

truncation

Set **truncation** equal to ON to indicate that the data of the report table must be reported with limited precision. Set **truncation** equal to OFF to indicate that the data of the report table must be reported with full precision.

If **truncation** is omitted, the data of the report table are reported with limited precision.

References:

- *Scripting Parametric Studies*

Additional Data for FILE and XYPLOT:

Required data

file

Set **file** equal to the name of the file to which the report table is to be written. The file name must be enclosed by matching quotation marks.

aStudy.sample()

Products: Abaqus/Standard Abaqus/Explicit

Command

```
aStudy.sample (token, additional data)
```

Tokens

INTERVAL	Use this token to sample a parameter at equal intervals.
NUMBER	Use this token to sample a given number of values of a parameter.
PRINT	Use this token to print parameter samples.
REFERENCE	Use this token to sample parameter values specified with respect to a reference parameter value.
VALUES	Use this token to sample particular values of a parameter.

References:

- *Scripting Parametric Studies*

Additional Data for INTERVAL:

Required Data

interval

Set **interval** equal to the sampling interval. For a continuous valued parameter, values are sampled at equally spaced intervals based on the numerical value of the parameter. For a discrete valued parameter, values are sampled at equally spaced intervals based on the indexing of the sequence of parameter values.

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

Optional Data

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, `(10., 20.)` or `[10., 20.]`. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, `(1., 2., 5., 3.)` or `[1., 2., 5., 3.]`.

If **domain** is specified in this command as well as in the define command for this parameter, the **domain** specification in this command is used for sampling.

If **domain** is omitted from this command, it must have been specified in the define command.

Additional Data for NUMBER:

Required Data

number

Set **number** equal to the number of equally spaced values to be sampled for the parameter.

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, 'par1'. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, ('par1', 'par2', 'par3') or ['par1', 'par2', 'par3'].

Optional Data

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, (10., 20.) or [10., 20.]. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, (1., 2., 5., 3.) or [1., 2., 5., 3.].

If **domain** is specified in this command as well as in the define command for this parameter, the **domain** specification in this command is used for sampling.

If **domain** is omitted from this command, it must have been specified in the define command.

Additional Data for PRINT:

Optional Data

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, 'par1'. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, ('par1', 'par2', 'par3') or ['par1', 'par2', 'par3'].

If **par** is omitted, parameter samplings are printed for all parameters in the parametric study.

Additional Data for REFERENCE:

Required Data

interval

Set **interval** equal to the sampling interval. For a continuous valued parameter, values are sampled at equally spaced intervals about the reference value based on the numerical value of the parameter. For a discrete valued parameter, values are sampled at equally spaced

intervals about the reference value based on the indexing of the sequence of parameter values.

numSymPairs

Set **numSymPairs** equal to the number of pairs of parameter values to be sampled symmetrically about the reference value of the parameter.

par

Set **par** equal to the name of the parameter being sampled. This name must be enclosed by matching quotation marks; for example, `'par1'`.

Optional Data

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, `(10., 20.)` or `[10., 20.]`. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, `(1., 2., 5., 3.)` or `[1., 2., 5., 3.]`.

If **domain** is specified in this command as well as in the `define` command for this parameter, the **domain** specification in this command is used for sampling.

In the case of a discrete valued parameter if **domain** is omitted from this command, it must have been specified in the `define` command.

reference

For a continuous valued parameter, set **reference** equal to the reference value of the parameter. For discrete valued parameters, set **reference** equal to the index in the sequence of parameter values; indexing starts at zero, so that the first value of the sequence corresponds to index zero and the last value of the sequence corresponds to an index equal to the number of values in the sequence minus one.

If **reference** is specified in this command as well as in the `define` command for this parameter, the **reference** specification in this command is used for sampling.

If **reference** is omitted from this command, it must have been specified in the `define` command.

Additional Data for VALUES:

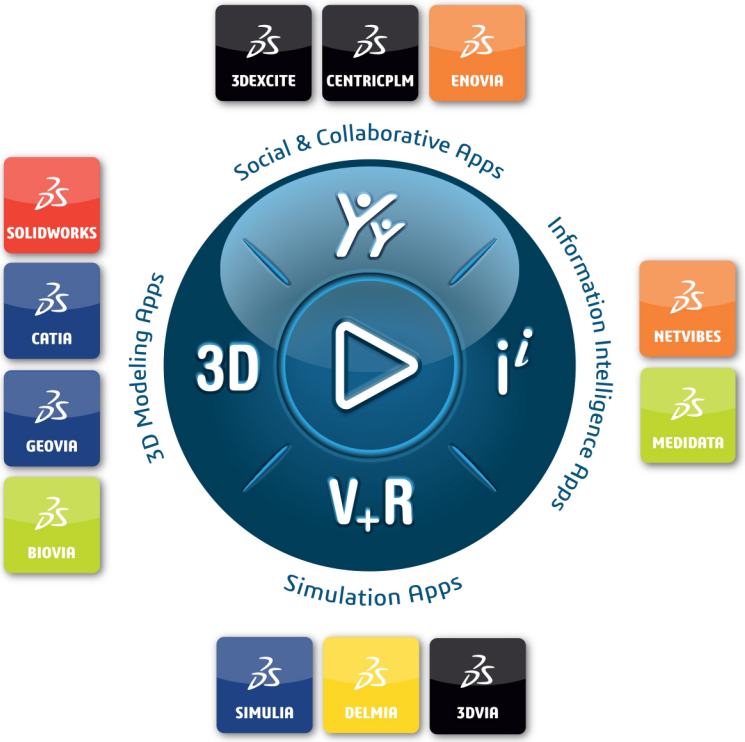
Required Data

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

values

Set **values** equal to the sequence of parameter values that constitute the sample. This sequence must be given inside parentheses or brackets and must contain values separated by commas; for example, `('CAX4', 'CAX4R', 'CAX4H')` or `[10., 20., 40.]`.



Our **3D**EXPERIENCE® platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes is a catalyst for human progress. We provide business and people with collaborative virtual environments to imagine sustainable innovations. By creating virtual twin experiences of the real world with our **3D**EXPERIENCE platform and applications, our customers can redefine the creation, production and life-cycle-management processes of their offer and thus have a meaningful impact to make the world more sustainable. The beauty of the Experience Economy is that it is a human-centered economy for the benefit of all – consumers, patients and citizens.

Dassault Systèmes brings value to more than 300,000 customers of all sizes, in all industries, in more than 150 countries. For more information, visit **www.3ds.com**.

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes
17F, Foxconn Building,
No. 1366, Lujiazui Ring Road
Pilot Free Trade Zone, Shanghai 200120
China

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

DASSAULT SYSTEMES

©2026 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries.