SIMULIA

3DEXPERIENCE®

ABAQUS 2025 FD04
GETTING STARTED WITH ABAQUS/CAE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....11
Getting Started with Abaqus/CAE....12
What's New....13
Introduction....14
The Abaqus products....15
Getting started with Abaqus....17
How to use this guide....18
Conventions used in this guide....19
Basic mouse actions....20
Abaqus documentation....21
Getting help....24
Support....25
Technical support....26
Systems support....27
Support for academic institutions....28
A quick review of the finite element method....29
Obtaining nodal displacements using implicit methods....30
Stress wave propagation illustrated....32
Abaqus Basics....34
Preprocessing, simulation, and postprocessing....35
Components of an Abaqus analysis model....36
Introduction to Abaqus/CAE....38
Starting Abaqus/CAE....39
Components of the main window....40
What is a module?....43
What is the Model Tree?....46
Example: creating a model of an overhead hoist....48
Overhead hoist analysis....49
Units....50
Creating a part....51
Creating a material....56
Defining and assigning section properties....57
Defining the assembly....60
Configuring your analysis....61
Applying boundary conditions and loads to the model....65
Meshing the model....70
Creating an analysis job....73
Checking the model....74
Running the analysis....76
Postprocessing....77
Rerunning the analysis using Abaqus/Explicit....89
Postprocessing the dynamic analysis results....94
Comparison of implicit and explicit procedures....96

Choosing between implicit and explicit analysis....97
Cost of mesh refinement in implicit and explicit analyses....98
Summary....99
Finite Elements and Rigid Bodies....100
Finite elements....101
Characterizing elements....102
Continuum elements....106
Shell elements....109
Beam elements....111
Truss elements....113
Rigid bodies....114
Determining when to use a rigid body....115
Components of a rigid body....117
Rigid elements....118
Summary....119
Using Continuum Elements....120
Element formulation and integration....121
About element formulation and integration....122
Full integration....123
Reduced integration....126
Incompatible mode elements....128
Hybrid elements....130
Selecting continuum elements....131
Example: connecting lug....132
Connecting lug analysis....133
Preprocessing—creating the model with Abaqus/CAE....134
Postprocessing—visualizing the results....150
Rerunning the analysis using Abaqus/Explicit....172
Postprocessing the dynamic analysis results....173
Mesh convergence....178
Related Abaqus examples....182
Suggested reading....183
Summary....184
Using Shell Elements....185
Element geometry....186
Shell thickness and section points....187
Shell normals and shell surfaces....188
Initial shell curvature....189
Reference surface offsets....190
Shell formulation – thick or thin....191
Shell material directions....192
Default local material directions....193
Creating alternative material directions....194
Selecting shell elements 195
Example: skew plate....196
Skew plate analysis....197
Preprocessing—creating the model with Abaqus/CAE....198

Postprocessing....204
Related Abaqus examples....212
Suggested reading....213
Summary....214
Using Beam Elements....215
Beam cross-section geometry....216
Beam cross-section analysis....217
Section points....218
Cross-section orientation....219
Beam element curvature....220
Nodal offsets in beam sections....221
Formulation and integration....222
Shear deformation....223
Torsional response—warping....224
Selecting beam elements....226
Example: cargo crane....227
Cargo crane analysis....228
Preprocessing—creating the model with Abaqus/CAE....230
Postprocessing....250
Related Abaqus examples....254
Suggested reading....255
Summary....256
Linear Dynamics....257
Introduction....258
Natural frequencies and mode shapes....259
Modal superposition....260
Damping....261
Definition of damping in Abaqus/Standard....262
Choosing damping values....263
Element selection....264
Mesh design for dynamics....265
Example: cargo crane under dynamic loading....266
Modifications to the model....267
Results....272
Postprocessing....276
Effect of the number of modes....282
Effect of damping....283
Comparison with direct time integration....284
Other dynamic procedures....286
Linear modal dynamics....287
Nonlinear dynamics....288
Related Abaqus examples....289
Suggested reading....290
Summary....291
Nonlinearity....292
Linear analyses versus nonlinear analyses....293
Sources of nonlinearity....294

Material nonlinearity....295
Boundary nonlinearity....296
Geometric nonlinearity....297
The solution of nonlinear problems....299
Nonlinear load-displacement curve....300
Steps, increments, and iterations....302
Equilibrium iterations and convergence in Abaqus/Standard....303
Automatic incrementation control in Abaqus/Standard....305
Including nonlinearity in an Abaqus analysis....306
Example: nonlinear skew plate....307
Modifications to the model....308
Job diagnostics....311
Postprocessing....316
Running the analysis in Abaqus/Explicit....322
Related Abaqus examples....323
Suggested reading....324
Summary....325
Nonlinear Explicit Dynamics....326
Types of problems suited for Abaqus/Explicit....327
Explicit dynamic finite element methods....328
Explicit time integration....329
Comparison of implicit and explicit time integration procedures....331
Advantages of the explicit time integration method....332
Automatic time incrementation and stability....333
Conditional stability of the explicit method....334
Definition of the stability limit....335
Fully automatic time incrementation versus fixed time incrementation in Abaqus/Explicit....336
Mass scaling to control time incrementation....337
Effect of material on stability limit....338
Effect of mesh on stability limit....339
Numerical instability....340
Example: stress wave propagation in a bar....341
Stress wave propagation analysis....342
Preprocessing—creating the model with Abaqus/CAE....343
Postprocessing....356
How the mesh affects the stable time increment and CPU time....361
How the material affects the stable time increment and CPU time....363
Damping of dynamic oscillations....364
Bulk viscosity....365
Viscous pressure....367
Material damping....368
Discrete dashpots....369
Energy balance....370
Statement of energy balance....371
Output of the energy balance....372
Summary....374
Materials....375
Defining materials in Abaqus....376

Plasticity in ductile metals....377
Characteristics of plasticity in ductile metals....378
Stress and strain measures for finite deformations....380
Defining plasticity in Abaqus....381
Selecting elements for elastic-plastic problems....386
Example: connecting lug with plasticity....387
Connecting lug with plasticity analysis....388
Modifications to the model....389
Job monitor and diagnostics....390
Postprocessing the results....395
Adding hardening to the material model....396
Running the analysis with plastic hardening....397
Postprocessing the results....398
Example: blast loading on a stiffened plate....405
Blast loading on a stiffened plate analysis....406
Preprocessing—creating the model with Abaqus/CAE....407
Postprocessing....414
Reviewing the analysis....421
Hyperelasticity....424
Introduction....425
Compressibility....426
Strain energy potential....428
Defining hyperelastic behavior using test data....429
Example: axisymmetric mount....431
Axisymmetric mount analysis....432
Symmetry....433
Preprocessing—creating the model with Abaqus/CAE....435
Postprocessing....447
Mesh design for large distortions....454
Techniques for reducing volumetric locking....456
Related Abaqus examples....457
Suggested reading....458
Summary....459
Multiple Step Analysis....460
General analysis procedures....461
Time in general analysis steps....462
Specifying loads in general steps....463
Linear perturbation analysis....464
Linear perturbation steps....465
Specifying loads in linear perturbation steps....466
Example: vibration of a piping system....469
Vibrational frequencies analysis....470
Preprocessing—creating the model with Abaqus/CAE....471
Monitoring the job....473
Postprocessing....474
Restart analysis....476
The restart and state files....477
Restarting an analysis....478

Example: restarting the pipe vibration analysis....480
Creating a restart analysis model....481
Monitoring the job....482
Postprocessing the restart analysis results....483
Related Abaqus examples....487
Summary....488
Contact....489
Contact capabilities in Abaqus....490
Defining surfaces....491
Interaction between surfaces....493
Behavior normal to the surfaces....494
Sliding of the surfaces....495
Friction models....496
Other contact interaction options....498
Surface-based constraints....499
Defining contact in Abaqus/Standard....500
Contact interactions....501
Secondary and main surfaces....502
Contact discretization....503
Small and finite sliding....504
Element selection....505
Contact algorithm....506
Modeling issues for rigid surfaces in Abaqus/Standard....507
Abaqus/Standard 2D example: forming a channel....509
Simulation of the forming of a channel....510
Preprocessing—creating the model with Abaqus/CAE....512
Job monitoring....521
Troubleshooting Abaqus/Standard contact analyses....523
Postprocessing....525
General contact in Abaqus/Standard....530
Abaqus/Standard 3D example: shearing of a lap joint....531
Simulation of the shearing of a lap joint....532
Preprocessing—creating the model with Abaqus/CAE....533
Postprocessing....543
Defining contact in Abaqus/Explicit....546
Modeling considerations in Abaqus/Explicit....548
Correct surface definitions....549
Overconstraining the model....555
Mesh refinement....556
Initial contact overclosure....557
Abaqus/Explicit example: circuit board drop test....558
Circuit board drop test analysis....559
Preprocessing—creating the model with Abaqus/CAE....560
Postprocessing....579
Rerunning the analysis with output filtering....587
Compatibility between Abaqus/Standard and Abaqus/Explicit....596
Related Abaqus examples....597
Suggested reading....598

Summary....599
Quasi-Static Analysis with Abaqus/Explicit....600
Analogy for explicit dynamics....601
Loading rates....602
Smooth amplitude curves....603
Structural problems....604
Metal forming problems....606
Mass scaling....607
Energy balance....608
Example: forming a channel in Abaqus/Explicit....610
Preprocessing—rerunning the model with Abaqus/Explicit....611
Forming analysis—attempt 2....619
Discussion of the two forming attempts....621
Methods of speeding up the analysis....625
Springback analysis in Abaqus/Standard....629
Summary....631
Example Files....632
Overhead hoist frame....633
Fetching the Python script....633
gsi_frame_caemodel.py....633
Connecting lug....637
Fetching the Python script....637
gsi_lug_caemodel.py....637
Skew plate....642
Fetching the Python script....642
gsi_skewplate_caemodel.py....642
Cargo crane....647
Fetching the Python script....647
gsi_crane_caemodel.py....647
Cargo crane – dynamic loading....657
Fetching the Python script....657
gsi_dyncrane_caemodel.py....657
Nonlinear skew plate....669
Fetching the Python script....669
gsi_nlskewplate_caemodel.py....669
Stress wave propagation in a bar....674
Fetching the Python script....674
gxi_stresswave_caemodel.py....674
Connecting lug with plasticity....677
Fetching the Python script....677
gsi_plasticlug_caemodel.py....677
Blast loading on a stiffened plate....682
Fetching the Python script....682
gxi_stiffplate_caemodel.py....682
Axisymmetric mount....687
Fetching the Python script....687
gsi_mount_caemodel.py....687
Vibration of a piping system....693

Fetching the Python script....693
gsi_pipe_caemodel.py....693
Forming a channel....698
Fetching the Python script....698
gsi_channel_caemodel.py....698
gsi_channel_caemodel_spring.py....708
Shearing of a lap joint....711
Fetching the Python script....711
gsi_lap_joint_caemodel.py....711
lap-joint-alum.txt....718
lap-joint-titanium.txt....718
Circuit board drop test....719
Fetching the Python script....719
gxi_circuit_caemodel.py....719
drop-test-foam.txt....727
Creating and Analyzing a Simple Model in Abaqus/CAE....728
Understanding Abaqus/CAE modules....729
Understanding the Model Tree....731
Creating a part....733
Creating a material....736
Defining and assigning section properties....737
Defining a homogeneous solid section....738
Assigning the section to the cantilever beam....739
Assembling the model....740
Defining your analysis steps....741
Creating an analysis step....742
Requesting data output....743
Applying a boundary condition and a load to the model....745
Applying a boundary condition to one end of the cantilever beam....746
Applying a load to the top of the cantilever beam....748
Meshing the model....749
Assigning mesh controls....750
Assigning an Abaqus element type....751
Creating the mesh....752
Creating and submitting an analysis job....753
Viewing the results of your analysis....754
Summary....756
Using Additional Techniques to Create and Analyze a Model in Abaqus/CAE....758
Creating an assembly composed of a hinge held together by a pin 759
Creating the first hinge piece....760
Creating the cube....761
Adding the flange to the base feature....763
Modifying a feature....768
Creating the sketch plane....769
Sketching the lubrication hole....772
Assigning section properties to the hinge part....776
Creating a material....777
Defining a section....778

Assigning the section....779
Creating and modifying a second hinge piece....780
Copying the hinge....781
Modifying the copy of the hinge....782
Creating the pin....783
Creating the pin....784
Assigning the rigid body reference point....785
Assembling the model....786
Creating an instance of the first hinge....787
Creating an instance of the second hinge....788
Positioning the solid hinge piece....789
Creating and positioning an instance of the pin....795
Defining analysis steps....798
Creating the analysis steps....799
Requesting output....800
Selecting a degree of freedom to monitor....802
Creating surfaces to use in contact interactions....803
Hide part instances in the assembly....803
Define a surface on the pin....803
Defining contact between regions of the model....804
Creating an interaction property....805
Defining a general contact interaction....806
Applying boundary conditions and loads to the assembly....807
Application of boundary conditions and load....808
Constraining the hinge piece with the lubrication hole....810
Constraining the pin....812
Modifying the boundary condition applied to the pin....813
Constraining the solid hinge piece....814
Applying a load to the solid hinge....815
Meshing the assembly....816
Deciding what needs to be partitioned....817
Partitioning the flange with the lubrication hole....819
Assigning mesh controls....821
Assigning the Abaqus element type....822
Seeding the part instances....823
Meshing the assembly....824
Creating and submitting a job....825
Viewing the results of your analysis....826
Displaying and customizing a contour plot....827
Using display groups....829
Summary....831
Viewing the Output from Your Analysis....832
Displaying output....833
Which variables are in the output database?....835
Reading the output database....837
Customizing a model plot....839
Displaying the deformed model shape....840
Displaying a deformed shape plot....841

Superimposing the undeformed shape on the deformed shape....843
Displaying and customizing a contour plot....844
Displaying a contour plot....845
Selecting the variable to plot....846
Customizing a contour plot....847
Animating a contour plot....849
Displaying and customizing a symbol plot....851
Displaying a vector symbol plot....852
Customizing the symbol plot....853
Displaying and customizing a material orientation plot....854
Displaying a material orientation plot....855
Customizing a material orientation plot....856
Displaying and customizing an X–Y plot....857
Displaying an X–Y plot....858
Customizing an X–Y plot....859
Operating on X–Y data....862
Creating the stress versus time and strain versus time data objects....863
Combining the data objects....864
Plotting and customizing the stress-strain curve....865
Probing an X–Y plot....866
Displaying results along a path....867
Creating a node list path....868
Displaying results along a node list path....869
Summary....870

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., 2025.

For a full list of the third-party software contained in this release, please go to the Legal Notices in the Abaqus 2025 HTML documentation, which can be obtained from a documentation installation, or in the SIMULIA Established Products 2025 Program Directory, which is available on *www.3ds.com*.

Getting Started with Abaqus/CAE

Getting Started with Abaqus/CAE is a self-paced tutorial designed to help new users become familiar with using Abaqus/CAE to create solid, shell, and beam and truss models; analyze these models in Abaqus/Standard and Abaqus/Explicit; and view the results in the Visualization module of Abaqus/CAE.
This guide contains a number of fully worked examples that provide practical guidelines for performing structural analyses with Abaqus. In addition, comprehensive tutorials are provided to introduce users familiar with the Abaqus solver products to the Abaqus/CAE interface.

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.

What's New

This page describes recent changes in Getting Started with Abaqus/CAE.

2025 FD01

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 *Getting Started with Abaqus/CAE*.

Introduction

Abaqus is a suite of powerful engineering simulation programs, based on the finite element method, that can solve problems ranging from relatively simple linear analyses to the most challenging nonlinear simulations. Abaqus contains an extensive library of elements that can model virtually any geometry. It has an equally extensive list of material models that can simulate the behavior of most typical engineering materials including metals, rubber, polymers, composites, reinforced concrete, crushable and resilient foams, and geotechnical materials such as soils and rock. Designed as a general-purpose simulation tool, Abaqus can be used to study more than just structural (stress/displacement) problems. It can simulate problems in such diverse areas as heat transfer, mass diffusion, thermal management of electrical components (coupled thermal-electrical analyses), acoustics, soil mechanics (coupled pore fluid-stress analyses), piezoelectric analysis, electromagnetic analysis, and fluid dynamics.

Abaqus offers a wide range of capabilities for simulation of linear and nonlinear applications. Problems with multiple components are modeled by associating the geometry defining each component with the appropriate material models and specifying component interactions. In a nonlinear analysis Abaqus automatically chooses appropriate load increments and convergence tolerances and continually adjusts them during the analysis to ensure that an accurate solution is obtained efficiently.

In this section:

- *The Abaqus products*
- *Getting started with Abaqus*
- *Abaqus documentation*
- *Getting help*
- *Support*
- *A quick review of the finite element method*

The Abaqus products

Abaqus consists of two main analysis products—Abaqus/Standard and Abaqus/Explicit. Several add-on analysis options are available to further extend the capabilities of Abaqus/Standard and Abaqus/Explicit. The Abaqus/Aqua option works with Abaqus/Standard and Abaqus/Explicit. The Abaqus/Design and Abaqus/AMS options work with Abaqus/Standard. Abaqus/Foundation is an optional subset of Abaqus/Standard. Abaqus/CAE is the Complete Abaqus Environment that includes capabilities for creating Abaqus models, interactively submitting and monitoring Abaqus jobs, and evaluating results. Abaqus/Viewer is a subset of Abaqus/CAE that includes just the postprocessing functionality. Abaqus also provides translators that convert geometry from third-party CAD systems to models for Abaqus/CAE, convert entities from third-party preprocessors to input for Abaqus analyses, and that convert output from Abaqus analyses to entities for third-party postprocessors.

Abaqus/Standard

Abaqus/Standard is a general-purpose analysis product that can solve a wide range of linear and nonlinear problems involving the static, dynamic, thermal, electrical, and electromagnetic response of components. This product is discussed in detail in this guide. Abaqus/Standard solves a system of equations implicitly at each solution "increment." In contrast, Abaqus/Explicit marches a solution forward through time in small time increments without solving a coupled system of equations at each increment (or even forming a global stiffness matrix).

Abaqus/Explicit

Abaqus/Explicit is a special-purpose analysis product that uses an explicit dynamic finite element formulation. It is suitable for modeling brief, transient dynamic events, such as impact and blast problems, and is also very efficient for highly nonlinear problems involving changing contact conditions, such as forming simulations. Abaqus/Explicit is discussed in detail in this guide.

Abaqus/CAE

Abaqus/CAE (Complete Abaqus Environment) is an interactive, graphical environment for Abaqus. It allows models to be created quickly and easily by producing or importing the geometry of the structure to be analyzed and decomposing the geometry into meshable regions. Physical and material properties can be assigned to the geometry, together with loads and boundary conditions. Abaqus/CAE contains very powerful options to mesh the geometry and to verify the resulting analysis model. Once the model is complete, Abaqus/CAE can submit, monitor, and control the analysis jobs. The Visualization module can then be used to interpret the results. Abaqus/CAE is discussed in this guide.

Abaqus/Viewer

Abaqus/Viewer is a subset of Abaqus/CAE that contains only the postprocessing capabilities of the Visualization module. The discussions of the Visualization module in this guide apply equally to Abaqus/Viewer.

Abaqus/Aqua

Abaqus/Aqua is a set of optional capabilities that can be added to Abaqus/Standard and Abaqus/Explicit. It is intended for the simulation of offshore structures, such as oil platforms. Some of the optional capabilities include the effects of wave and wind loading and buoyancy. Abaqus/Aqua is not discussed in this guide.

Abaqus/Design

Abaqus/Design is a set of optional capabilities that can be added to Abaqus/Standard to perform design sensitivity calculations. Abaqus/Design is not discussed in this guide.

Abaqus/AMS

Abaqus/AMS is an optional capability that can be added to Abaqus/Standard. It uses the automatic multi-level substructuring (AMS) eigensolver during a natural frequency extraction. Abaqus/AMS is not discussed in this guide.

Abaqus/Foundation

Abaqus/Foundation offers more efficient access to the linear static and dynamic analysis functionality in Abaqus/Standard. Abaqus/Foundation is not discussed in this guide.

Geometry translators

Abaqus provides the following translators for converting geometry from third-party CAD systems to parts and assemblies for Abaqus/CAE:

- The SIMULIA Associative Interface for Abaqus/CAE creates a link between CATIA V6 and Abaqus/CAE that allows you to transfer model data and propagate design changes from CATIA V6 to Abaqus/CAE.
- The CATIA V5 Associative Interface creates a link between CATIA V5 and Abaqus/CAE that allows you to transfer model data and propagate design changes from CATIA V5 to Abaqus/CAE.
- The SOLIDWORKS Associative Interface creates a link between SOLIDWORKS and Abaqus/CAE that allows you to transfer model data and propagate design changes from SOLIDWORKS to Abaqus/CAE.
- The Pro/ENGINEER Associative Interface creates a link between Pro/ENGINEER and Abaqus/CAE that allows you to transfer model data and propagate design changes between Pro/ENGINEER and Abaqus/CAE.
- The Geometry Translator for Parasolid allows you to import the geometry of Parasolid-format parts and assemblies directly into Abaqus/CAE.

In addition, the Abaqus/CAE Associative Interface for NX creates a link between NX and Abaqus/CAE that allows you to transfer model data and propagate design changes between NX and Abaqus/CAE. The Abaqus/CAE Associative Interface for NX can be purchased and downloaded from Elysium Inc. (*http://www.elysiuminc.com*).

The geometry translators are not discussed in this guide.

Translator utilities

Abaqus provides the translators for converting entities from third-party preprocessors to input for Abaqus analyses or for converting output from Abaqus analyses to entities for third-party postprocessors.

See *Execution Procedures* for more information. The translator utilities are not discussed in this guide.

Getting started with Abaqus

This guide is an introductory text designed to give new users guidance in creating solid, shell, beam, and truss models with Abaqus/CAE, analyzing these models with Abaqus/Standard and Abaqus/Explicit, and viewing the results in the Visualization module.
You do not need any previous knowledge of Abaqus to benefit from this guide, although some previous exposure to the finite element method is recommended. If you are already familiar with the Abaqus solver products (Abaqus/Standard or Abaqus/Explicit) but would like an introduction to the Abaqus/CAE interface, three tutorials are provided in the appendices of the HTML version of this guide to lead you through the modeling process in Abaqus/CAE.

This document covers primarily stress/displacement simulations, concentrating on both linear and nonlinear static analyses as well as dynamic analyses. Other types of simulations, such as heat transfer and mass diffusion, are not covered.

In this section:

- *How to use this guide*
- *Conventions used in this guide*
- *Basic mouse actions*

How to use this guide

The different sections of this guide are addressed to different types of users.

Tutorials for new Abaqus users

If you are completely new to Abaqus, we recommend that you follow each of the self-paced tutorials in this guide. Each of the sections in this guide introduces one or more topics relevant to using Abaqus/Standard and Abaqus/Explicit. Throughout the guide the term Abaqus is used to refer to both of these analysis products; the individual product names are used when information applies to only one product. Most sections contain a short discussion of the topic or topics being considered and one or two tutorial examples. You should work through the examples carefully since they contain a great deal of practical advice on using Abaqus.

The capabilities of Abaqus/CAE are introduced gradually in these examples. It is assumed that you will use Abaqus/CAE to create the models used in the examples. You can also generate the model for any example using a script that replicates the complete analysis model for the problem. A model created from a script may differ slightly from that created by following the steps in this guide. These differences, such as material names or node numbers, are minor and can be ignored. Scripts are available in two locations:

- A Python script is provided for each example in *Example Files*. The same section also provides instructions on how to fetch the script and run it within Abaqus/CAE.
- Abaqus/CAE plug-in scripts are provided for each example in the **Getting Started Examples** dialog box of the Abaqus/CAE Plug-in toolset. For more information about running these scripts, see *Running the Getting Started with Abaqus examples*.

This section is a short introduction to Abaqus and this guide. *Abaqus Basics*, which is centered around a simple example, covers the basics of using Abaqus. By the end of *Abaqus Basics*, you will know the fundamentals of how to prepare a model for an Abaqus simulation, check the data, run the analysis job, and view the results.

Finite Elements and Rigid Bodies presents an overview of the main element families available in Abaqus. The use of continuum (solid) elements, shell elements, and beam elements is discussed in *Using Continuum Elements*, *Using Shell Elements*, and *Using Beam Elements*, respectively.

Linear dynamic analyses are discussed in *Linear Dynamics*. *Nonlinearity* introduces the concept of nonlinearity in general, and geometric nonlinearity in particular, and contains the first nonlinear Abaqus simulation. Nonlinear dynamic analyses are discussed in *Nonlinear Explicit Dynamics*, and material nonlinearity is introduced in *Materials*. *Multiple Step Analysis* introduces the concept of multistep simulations, and *Contact* discusses the many issues that arise in contact analyses. Using Abaqus/Explicit to solve quasi-static problems is presented in *Quasi-Static Analysis with Abaqus/Explicit*. The illustrative example is a sheet metal forming simulation, which requires importing between Abaqus/Explicit and Abaqus/Standard to perform the forming and springback analyses efficiently.

Abaqus/CAE tutorials for experienced Abaqus users

Several tutorials are provided in the HTML version of this guide to introduce users familiar with the Abaqus analysis products to the Abaqus/CAE interface. In *Creating and Analyzing a Simple Model in Abaqus/CAE*, you create a simple model, analyze it, and then view the results. The second tutorial, *Using Additional Techniques to Create and Analyze a Model in Abaqus/CAE* is more complex and illustrates how parts, sketches, datum geometry, and partitions work together and how you assemble part instances. *Viewing the Output from Your Analysis* demonstrates how you can use the Visualization module (also licensed separately as Abaqus/Viewer) to display your results in a variety of formats and how you can customize the display.

Conventions used in this guide

This guide adheres to the following conventions:

Typographical conventions

Different text styles are used in the tutorial examples to indicate specific actions or identify items.

- Input in `COURIER FONT` should be typed into Abaqus/CAE or your computer exactly as shown. For example,

  ```
  abaqus cae
  ```

  would be typed on your computer to run Abaqus/CAE.
- Menu selections, tabs within dialog boxes, and labels of items on the screen in Abaqus/CAE are indicated in bold:

  View->Graphics Options
  Contour Plot Options

Sketcher figures

Sketches are two-dimensional profiles that form the geometry of features defining an Abaqus/CAE native part. You use the Sketcher to create these sketches, as shown in *Figure 1*. The Sketcher displays major gridlines in a solid line style and the *X*- and *Y*-axes of the sketch and minor gridlines in a dashed line style. In order to visually distinguish the part sketch from the Sketcher grid, the gridlines in most of the Sketcher figures in this guide are dashed.

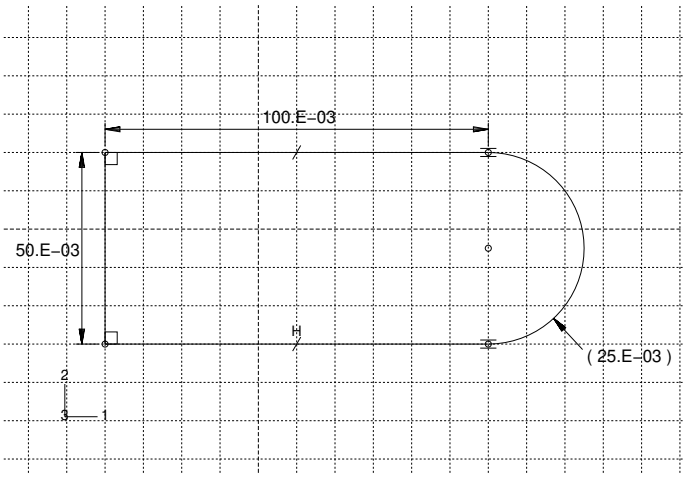


Figure 1: The Sketcher.

View orientation triad

By default, Abaqus/CAE uses the alphabetical option, *x*-*y*-*z*, for labeling the view orientation triad. In general, this guide adopts the numerical option, 1-2-3, to permit direct correspondence with degree of freedom and output labeling. The view orientation triad is shown in the lower left corner of *Figure 1*.

Basic mouse actions

Figure 1 shows the mouse button orientation for a left-handed and a right-handed 3-button mouse.

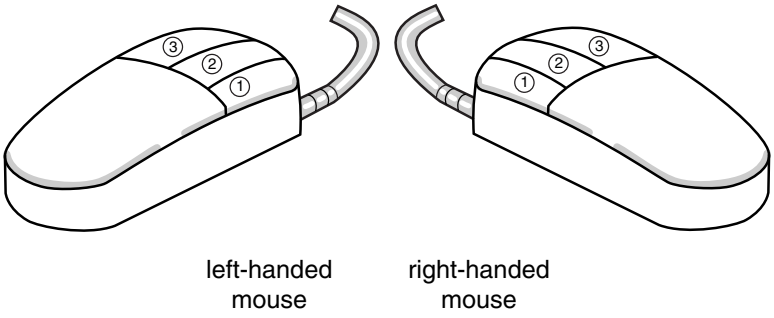


Figure 1: Mouse buttons.

The following terms describe actions you perform using the mouse:

Click

Press and quickly release the mouse button. Unless otherwise specified, the instruction “click” means that you should click mouse button 1.

Drag

Press and hold down mouse button 1 while moving the mouse.

Point

Move the mouse until the cursor is over the desired item.

Select

Point to an item and then click mouse button 1.

[Shift] + Click

Press and hold the **[Shift]** key, click mouse button 1, and then release the **[Shift]** key.

[Ctrl] + Click

Press and hold the **[Ctrl]** key, click mouse button 1, and then release the **[Ctrl]** key.

Abaqus/CAE is designed for use with a 3-button mouse. Accordingly, this guide refers to mouse buttons 1, 2, and 3 as shown in *Figure 1*. However, you can use Abaqus/CAE with a 2-button mouse as follows:

- The two mouse buttons are equivalent to mouse buttons 1 and 3 on a 3-button mouse.
- Pressing both mouse buttons simultaneously is equivalent to pressing mouse button 2 on a 3-button mouse.



Tip: You are instructed to click mouse button 2 in procedures throughout this guide. Make sure that you configure mouse button 2 (or the wheel button) to act as a middle button click.

Abaqus documentation

The documentation for Abaqus is extensive and complete.

The following documentation and publications are available from SIMULIA.

The following guides contain a complete description of the elements, material models, procedures, input specifications, etc. They are the basic guides for Abaqus/Standard and Abaqus/Explicit; and they provide both input file usage and Abaqus/CAE usage information. This guide regularly refers to these guides, so you should have them available as you work through the examples.

- *Abaqus Analysis Guide*
- *Abaqus Constraints Guide*
- *Abaqus Elements Guide*
- *Abaqus Execution Guide*
- *Abaqus Interactions Guide*
- *Abaqus Introduction & Spatial Modeling Guide*
- *Abaqus Materials Guide*
- *Abaqus Output Guide*
- *Abaqus Prescribed Conditions Guide*

The *Abaqus/CAE User's Guide* includes detailed descriptions of how to use Abaqus/CAE for model generation, analysis, and results evaluation and visualization. Abaqus/Viewer users should refer to the information on the Visualization module in this guide.

Other Abaqus documentation:

Abaqus Example Problems Guide

This guide contains detailed examples from which users can learn how to run simulations involving nontrivial physics. Many of the examples are worked with several different element types, mesh densities, and other variations. Cases include large motion of an elastic-plastic pipe hitting a rigid wall; inelastic buckling collapse of a thin-walled elbow; explosive loading of an elastic, viscoplastic thin ring; consolidation under a footing; buckling of a composite shell with a hole; and deep drawing of a metal sheet. It is generally useful to look for relevant examples in this guide and to review them when embarking on a new class of problem.

When you want to use a feature that you have not used before, you should look up one or more examples that use that feature. Then, use the example to familiarize yourself with the correct usage of the capability. To find an example that uses a certain feature, search the online documentation or use the **abaqus findkeyword** utility (see *Querying the Keyword/Problem Database* for more information).

All the input files associated with the examples are provided as part of the Abaqus installation. The **abaqus fetch** utility is used to extract sample Abaqus input files from the compressed archive files provided with the release (see *Fetching Sample Input Files* for more information). You can fetch any of the example files so that you can run the simulations yourself and review the results. You can also access the input files through the hyperlinks in the *Abaqus Example Problems Guide*.

Abaqus Benchmarks Guide

This guide contains benchmark problems that provide evidence that the software can reproduce a result from a benchmark defined by an external body or institution such as NAFEMS. The problems are sufficient to show accuracy and convergence compared to the benchmark data. The NAFEMS benchmark problems are included in this guide.

Abaqus Verification Guide

This guide contains test cases that provide evidence that the implementation of the numerical model produces the expected results for one or several well-defined options in the code. It may be useful to run these problems when learning to use a new capability. In addition, the supplied input data files provide good starting points to check the behavior of elements, materials, etc.

Abaqus Theory Guide

This guide contains detailed, precise discussions of all theoretical aspects of Abaqus. It is written to be understood by users with an engineering background.

Abaqus Keywords Guide

This guide contains a complete description of all the input options that are available in Abaqus/Standard and Abaqus/Explicit.

Abaqus User Subroutines Guide

This guide contains a complete description of all the user subroutines available for use in Abaqus analyses. It also discusses the utility routines that can be used when writing user subroutines.

Abaqus Release Notes

This guide contains brief descriptions of the new features available in the latest release of the Abaqus product line.

Abaqus Configuration Guide

This guide describes how to configure the installation for particular circumstances. Some of this information, of most relevance to users, is also provided in the *Abaqus Execution Guide*.

In addition to the documentation listed above, the following guides are available for Abaqus interfaces and custom programming techniques not discussed in this guide:

- *Abaqus Scripting User's Guide*
- *Abaqus Scripting Reference Guide*
- *Abaqus GUI Toolkit User's Guide*
- *Abaqus GUI Toolkit Reference Guide*

SIMULIA also provides documentation for all of the geometry translators described in *The Abaqus products*.

Additional publications available from SIMULIA:

Quality Assurance Manual

This document describes the Quality Management System followed by SIMULIA. It is a controlled document, provided to customers who subscribe to the Quality Monitoring Service. Contact your support office for more information.

Abaqus online resources

SIMULIA has a home page on the World Wide Web (*www.3ds.com/simulia*), containing a variety of useful information about the Abaqus suite of programs, including:

- Frequently asked questions
- Abaqus systems information and machine requirements
- Benchmark timing documents
- Error status reports
- Training course schedule
- Newsletters

Getting help

You may want to read additional information about Abaqus/CAE features at various points during the tutorials. The context-sensitive help system allows you to locate relevant information quickly and easily. Context-sensitive help is available for every item in the main window and in all dialog boxes.



Note:

- On Windows platforms, the help system uses your default web browser to display the online documentation.
- On Linux platforms, the help system searches the system path for Firefox. If the help system cannot find Firefox, an error is displayed.

  The **browser_type** and **browser_path** variables can be set in the environment file to modify this behavior. For more information, see *System customization parameters*.

1. From the main menu bar, select **Help->On Context**.

   Tip: You can also click the help tool  to access context-sensitive help.

   The cursor changes to a question mark.
2. Click any part of the main window except its frame.

   The online documentation opens in your web browser. Information about the item you selected is displayed.
3. Use hyperlinks and expand the table of contents to identify additional topics of interest.

Support

Both technical software support (for problems with creating a model or performing an analysis) and systems support (for installation, licensing, and hardware-related problems) for SIMULIA products are offered through a global network of support offices, as well as through our online support system.
Contact information for our regional offices is accessible from **SIMULIA > Services & Support** at *http://www.3ds.com/simulia*.

We welcome any suggestions for improvements to SIMULIA software, the support tool, or documentation. We will ensure that any enhancement requests you make are considered for future releases. If you wish to make a suggestion about the service or products, please contact your dedicated VAR, call your *Customer Support Center*, or *submit a request* directly through your DS Support Portal (only for customers)

In this section:

- *Technical support*
- *Systems support*
- *Support for academic institutions*

Technical support

SIMULIA technical support engineers are available to assist in clarifying Dassault Systèmes product features and checking errors by giving both general information on using Dassault Systèmes products and information on their application to specific analyses. If you have concerns about a model, analysis, or workflow, we suggest that you contact us at an early stage since it is usually easier to solve problems at the beginning of a project rather than trying to correct them at the end.

To access support, you can select the **Submit a request** link at *http://www.3ds.com/support* to see the online support tool or obtain support office contact information from the **LOCATION DIRECTORY** at *http://www.3ds.com/products-services/simulia/locations* to request support by telephone.

Please have the following information ready before calling the technical software support hotline, and include it in any written contacts:

- Your site identifier.
- The release of the Dassault Systèmes product that you are using.
- The type of computer and operating system that you are using. If you are using a cluster, make sure you included relevant details of the compute cluster. If you are having problems with servers when running the **3D**EXPERIENCE platform, make sure you have the details about these servers.
- The symptoms of any problems, including the exact error messages, if any.
- Workarounds or tests that you have already tried and the results of these tests.

When calling for support about a specific problem, any available files or messages may be helpful in answering questions that the support engineer may ask you.

The support engineer will try to diagnose your problem from the model description and a description of the difficulties you are having. The more detailed information you provide, the easier it will be for the support engineer to understand and solve your problem.

If the support engineer cannot diagnose your problem from this information, you may be asked to supply the model information data. The data can be attached to a support request in the online system. It can also be sent by means of email, or the support engineer can determine how you can transfer the data electronically to Dassault Systèmes.

If you are contacting us via telephone to discuss an existing problem, please give the receptionist the support engineer's name. If you are contacting us via email, please include the support engineer's name at the top of any email correspondence. If you are contacting us online (preferred for written communication), update the existing support request for the problem.

Systems support

Systems support engineers can help you resolve issues related to the installation and running of SIMULIA products, including licensing difficulties, that are not covered by technical software support

You should install Abaqus by carefully following the instructions in the *Abaqus Configuration Guide*. If you encounter problems with the installation or licensing, first review the instructions in the *Abaqus Configuration Guide* to ensure that they have been followed correctly. If this method does not resolve the problems, search for known installation problems in the Dassault Systèmes Knowledge Base by using the **Search our Knowledge** option at *http://www.3ds.com/support*. If this method does not address your situation, support is available by selecting the **Submit a request** link at *http://www.3ds.com/support* or from regional support offices. Contact information for our regional offices is accessible from **SIMULIA > Services & Support** at *http://www.3ds.com/simulia*.

Support for academic institutions

Under the terms of the Academic Software License Agreement we do not provide support to users at academic institutions apart from an initial period to cover resolving licensing and installation issues.

Academics should become part of the SIMULIA Learning Community at *https://www.3ds.com/products-services/simulia/simulia-academic-program/learning-community/*. Academic users can purchase a technical support account to coincide with their annual software lease. For more information, please contact your support office.

A quick review of the finite element method

This section reviews the basics of the finite element method. The first step of any finite element simulation is to *discretize* the actual geometry of the structure using a collection of *finite elements*.
Each finite element represents a discrete portion of the physical structure. The finite elements are joined by shared *nodes*. The collection of nodes and finite elements is called the *mesh*. The number of elements per unit of length, area, or in a mesh is referred to as the *mesh density*. In a stress analysis the displacements of the nodes are the fundamental variables that Abaqus calculates. Once the nodal displacements are known, the stresses and strains in each finite element can be determined easily.

In this section:

- *Obtaining nodal displacements using implicit methods*
- *Stress wave propagation illustrated*

Obtaining nodal displacements using implicit methods

A simple example of a truss, constrained at one end and loaded at the other end as shown in *Figure 1*, is used to introduce some terms and conventions used in this document.

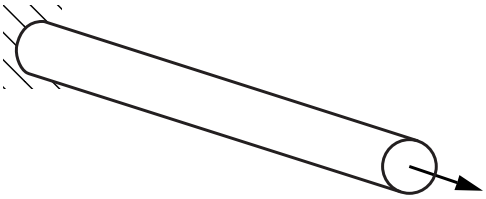


Figure 1: Truss problem.

The objective of the analysis is to find the displacement of the free end of the truss, the stress in the truss, and the reaction force at the constrained end of the truss.

In this case the rod shown in *Figure 1* will be modeled with two truss elements. In Abaqus truss elements can carry axial loads only. The discretized model is shown in *Figure 2* together with the node and element labels.

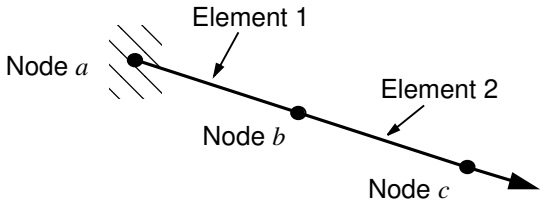


Figure 2: Discretized model of the truss problem.

Free-body diagrams for each node in the model are shown in *Figure 3*. In general each node will carry an external load applied to the model, P, and internal loads, I, caused by stresses in the elements attached to that node. For a model to be in static equilibrium, the net force acting on each node must be zero; i.e., the internal and external loads at each node must balance each other. For node a this equilibrium equation can be obtained as follows.

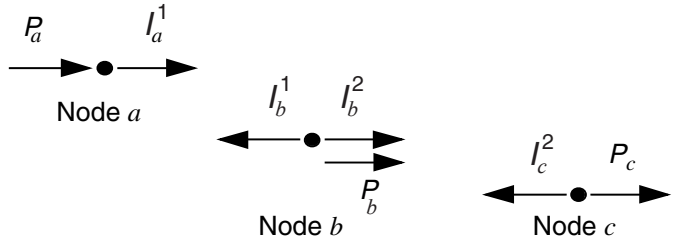


Figure 3: Free-body diagram for each node.

Assuming that the change in length of the rod is small, the strain in element 1 is given by

$$\varepsilon_{11} = \frac{u^b - u^a}{L},$$

where u^a and u^b are the displacements at nodes a and b, respectively, and L is the original length of the element.

Assuming that the material is linear elastic, the stress in the rod is given by the strain multiplied by the Young's modulus, E:

$$\sigma_{11} = E\varepsilon_{11}.$$

The axial force acting on the end node is equivalent to the stress in the rod multiplied by its cross-sectional area, *A*. Thus, a relationship between internal force, material properties, and displacements is obtained:

$$I_a^1 = \sigma_{11} A = E\varepsilon_{11} A = \frac{EA}{L}\left(u^b - u^a\right).$$

Equilibrium at node *a* can, therefore, be written as

$$P_a + \frac{EA}{L}\left(u^b - u^a\right) = 0.$$

Equilibrium at node *b* must take into account the internal forces acting from both elements joined at that node. The internal force from element 1 is now acting in the opposite direction and so becomes negative. The resulting equation is

$$P_b - \frac{EA}{L}\left(u^b - u^a\right) + \frac{EA}{L}\left(u^c - u^b\right) = 0.$$

For node *c* the equilibrium equation is

$$P_c - \frac{EA}{L}\left(u^c - u^b\right) = 0.$$

For implicit methods, the equilibrium equations need to be solved simultaneously to obtain the displacements of all the nodes. This requirement is best achieved by matrix techniques; therefore, write the internal and external force contributions as matrices. If the properties and dimensions of the two elements are the same, the equilibrium equations can be simplified as follows:

$$\begin{Bmatrix} P_a \\ P_b \\ P_c \end{Bmatrix} - \left(\frac{EA}{L}\right) \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{Bmatrix} u^a \\ u^b \\ u^c \end{Bmatrix} = 0.$$

In general, it may be that the element stiffnesses, the EA/L terms, are different from element to element; therefore, write the element stiffnesses as K_1 and K_2 for the two elements in the model. We are interested in obtaining the solution to the equilibrium equation in which the externally applied forces, *P*, are in equilibrium with the internally generated forces, *I*. When discussing this equation with reference to convergence and nonlinearity, we write it as

$$\{P\} - \{I\} = 0.$$

For the complete two-element, three-node structure we, therefore, modify the signs and rewrite the equilibrium equation as

$$\begin{Bmatrix} P_a \\ P_b \\ P_c \end{Bmatrix} - \begin{bmatrix} K_1 & -K_1 & 0 \\ -K_1 & (K_1 + K_2) & -K_2 \\ 0 & -K_2 & K_2 \end{bmatrix} \begin{Bmatrix} u^a \\ u^b \\ u^c \end{Bmatrix} = 0.$$

In an implicit method, such as that used in Abaqus/Standard, this system of equations can then be solved to obtain values for the three unknown variables: u^b, u^c, and P_a (u^a is specified in the problem as 0.0). Once the displacements are known, we can go back and use them to calculate the stresses in the truss elements. Implicit finite element methods require that a system of equations is solved at the end of each solution increment.

In contrast to implicit methods, an explicit method, such as that used in Abaqus/Explicit, does not require the solving of a simultaneous system of equations or the calculation of a global stiffness matrix. Instead, the solution is advanced kinematically from one increment to the next. The extension of the finite element method to explicit dynamics is covered in the following section.

Stress wave propagation illustrated

This section attempts to provide some conceptual understanding of how forces propagate through a model when using the explicit dynamics method. In this illustrative example we consider the propagation of a stress wave along a rod modeled with three elements, as shown in *Figure 1*. We will study the state of the rod as we increment through time.

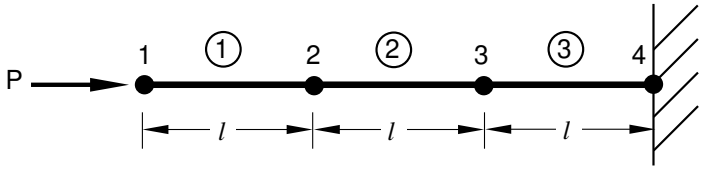
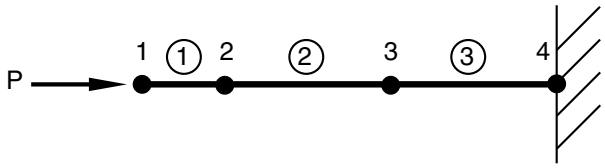


Figure 1: Initial configuration of a rod with a concentrated load, P, at the free end.

In the first time increment node 1 has an acceleration, $\ddot{u}_1$, as a result of the concentrated force, P, applied to it. The acceleration causes node 1 to have a velocity, $\dot{u}_1$, which, in turn, causes a strain rate, $\dot{\varepsilon}_{el1}$, in element 1. The increment of strain, $\Delta\varepsilon_{el1}$, in element 1 is obtained by integrating the strain rate through the time of increment 1. The total strain, ε_{el1}, is the sum of the initial strain, ε_0, and the increment in strain. In this case the initial strain is zero. Once the element strain has been calculated, the element stress, σ_{el1}, is obtained by applying the material constitutive model. For a linear elastic material the stress is simply the elastic modulus times the total strain. This process is shown in *Figure 2*. Nodes 2 and 3 do not move in the first increment since no force is applied to them.

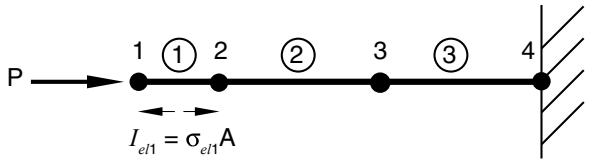


$$\ddot{u}_1 = \frac{P}{M_1} \Rightarrow \dot{u}_1 = \int \ddot{u}_1 dt \Rightarrow \dot{\varepsilon}_{el1} = \frac{-\dot{u}_1}{l} \Rightarrow \Delta\varepsilon_{el1} = \int \dot{\varepsilon}_{el1} dt$$

$$\Rightarrow \varepsilon_{el1} = \varepsilon_o + \Delta\varepsilon_{el1} \Rightarrow \sigma_{el1} = E\varepsilon_{el1}$$

Figure 2: Configuration at the end of increment 1 of a rod with a concentrated load, P, at the free end.

In the second increment the stresses in element 1 apply internal, element forces to the nodes associated with element 1, as shown in *Figure 3*. These element stresses are then used to calculate dynamic equilibrium at nodes 1 and 2.



$$\ddot{u}_1 = \frac{P - I_{el1}}{M_1} \Rightarrow \dot{u}_1 = \dot{u}_1^{old} + \int \ddot{u}_1 dt \qquad \dot{\varepsilon}_{el1} = \frac{\dot{u}_2 - \dot{u}_1}{l} \Rightarrow \Delta\varepsilon_{el1} = \int \dot{\varepsilon}_{el1} dt$$

$$\ddot{u}_2 = \frac{I_{el1}}{M_2} \Rightarrow \dot{u}_2 = \int \ddot{u}_2 dt \qquad \Rightarrow \varepsilon_{el1} = \varepsilon_{el1}^{old} + \Delta\varepsilon_{el1}$$

$$\Rightarrow \sigma_{el1} = E\varepsilon_{el1}$$

Figure 3: Configuration of the rod at the beginning of increment 2.

The process continues so that at the start of the third increment there are stresses in both elements 1 and 2, and there are forces at nodes 1, 2, and 3, as shown in *Figure 4*. The process continues until the analysis reaches the desired total time.

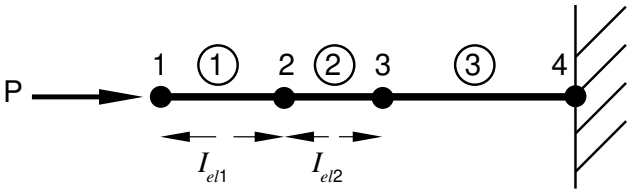


Figure 4: Configuration of the rod at the beginning of increment 3.

Abaqus Basics

This section describes the basic concepts of performing an Abaqus analysis.

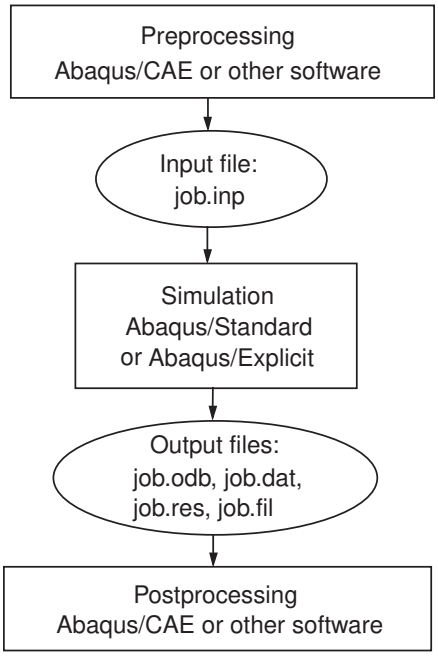
In this section:

- *Preprocessing, simulation, and postprocessing*
- *Components of an Abaqus analysis model*
- *Introduction to Abaqus/CAE*
- *Example: creating a model of an overhead hoist*
- *Comparison of implicit and explicit procedures*
- *Summary*

Preprocessing, simulation, and postprocessing

A complete Abaqus analysis usually consists of three distinct stages: preprocessing, simulation, and postprocessing.

These three stages are linked together by files as shown below:



Preprocessing (Abaqus/CAE)

In this stage you must define the model of the physical problem and create an Abaqus input file. The model is usually created graphically using Abaqus/CAE or another preprocessor, although the Abaqus input file for a simple analysis can be created directly using a text editor.

Simulation (Abaqus/Standard or Abaqus/Explicit)

The simulation, which normally is run as a background process, is the stage in which Abaqus/Standard or Abaqus/Explicit solves the numerical problem defined in the model. Examples of output from a stress analysis include displacements and stresses that are stored in binary files ready for postprocessing. Depending on the complexity of the problem being analyzed and the power of the computer being used, it may take anywhere from seconds to days to complete an analysis run.

Postprocessing (Abaqus/CAE)

You can evaluate the results once the simulation has been completed and the displacements, stresses, or other fundamental variables have been calculated. The evaluation is generally done interactively using the Visualization module of Abaqus/CAE or another postprocessor. The Visualization module, which reads the neutral binary output database file, has a variety of options for displaying the results, including color contour plots, animations, deformed shape plots, and *X*–*Y* plots.

Components of an Abaqus analysis model

An Abaqus model is composed of several different components that together describe the physical problem to be analyzed and the results to be obtained. At a minimum the analysis model consists of the following information: discretized geometry, element section properties, material data, loads and boundary conditions, analysis type, and output requests. The discussion in this chapter focuses on structural applications. Similar concepts apply for fluid dynamics.

Discretized geometry

Finite elements and nodes define the basic geometry of the physical structure being modeled in Abaqus. Each element in the model represents a discrete portion of the physical structure, which is, in turn, represented by many interconnected elements. Elements are connected to one another by shared nodes. The coordinates of the nodes and the connectivity of the elements—that is, which nodes belong to which elements—comprise the model geometry. The collection of all the elements and nodes in a model is called the *mesh*. Generally, the mesh will be only an approximation of the actual geometry of the structure.

The element type, shape, and location, as well as the overall number of elements used in the mesh, affect the results obtained from a simulation. The greater the mesh density (i.e., the greater the number of elements in the mesh), the more accurate the results. As the mesh density increases, the analysis results converge to a unique solution, and the computer time required for the analysis increases. The solution obtained from the numerical model is generally an approximation to the solution of the physical problem being simulated. The extent of the approximations made in the model's geometry, material behavior, boundary conditions, and loading determines how well the numerical simulation matches the physical problem.

Element section properties

Abaqus has a wide range of elements, many of which have geometry not defined completely by the coordinates of their nodes. For example, the layers of a composite shell or the dimensions of an I-beam section are not defined by the nodes of the element. Such additional geometric data are defined as physical properties of the element and are necessary to define the model geometry completely (see *Finite Elements and Rigid Bodies*).

Material data

Material properties for all elements must be specified. While high-quality material data are often difficult to obtain, particularly for the more complex material models, the validity of the Abaqus results is limited by the accuracy and extent of the material data.

Loads and boundary conditions

Loads distort the physical structure and, thus, create stress in it. The most common forms of loading include:

- point loads;
- pressure loads on surfaces;
- distributed tractions on surfaces;
- distributed edge loads and moments on shell edges;
- body forces, such as the force of gravity; and
- thermal loads.

Boundary conditions are used to constrain portions of the model to remain fixed (zero displacements) or to move by a prescribed amount (nonzero displacements).

In a static analysis enough boundary conditions must be used to prevent the model from moving as a rigid body in any direction; otherwise, unrestrained rigid body motion causes the stiffness matrix to be singular. A solver problem will occur during the solution stage and may cause the simulation to stop prematurely. Abaqus/Standard will issue a warning message if it detects a solver problem during a simulation. It is important that you learn to interpret such error messages. If you see a "numerical singularity" or "zero pivot" warning message during a static stress analysis, you should check whether all or part of your model lacks constraints against rigid body translations or rotations. Rigid body motions can consist of both translations and rotations of the components. The potential rigid body motions depend on the dimensionality of the model.

Dimensionality	**Possible Rigid Body Motion**
Three-dimensional	Translation in the 1-, 2-, and 3-directions.
	Rotation about the 1-, 2-, and 3-axes.
Axisymmetric	Translation in the 2-direction.
	Rotation about the 3-axis (axisymmetric rigid bodies only).
Plane stress, plane strain	Translation in the 1- and 2-directions.
	Rotation about the 3-axis.

By default, the 1-, 2-, and 3-directions are aligned with the axes of a global Cartesian coordinate system (discussed later).

In a dynamic analysis inertia forces prevent the model from undergoing infinite motion instantaneously as long as all separate parts in the model have some mass; therefore, solver problem warnings in a dynamic analysis usually indicate some other modeling problem, such as excessive plasticity.

Analysis type

Abaqus can carry out many different types of simulations, but this guide only covers the two most common: static and dynamic stress analyses.

In a static analysis the long-term response of the structure to the applied loads is obtained. In other cases the dynamic response of a structure to the loads may be of interest: for example, the effect of a sudden load on a component, such as occurs during an impact, or the response of a building in an earthquake.

Output requests

An Abaqus simulation can generate a large amount of output. To avoid using excessive disk space, you can limit the output to that required for interpreting the results.

Generally a preprocessor such as Abaqus/CAE is used to define the necessary components of the model.

Introduction to Abaqus/CAE

Abaqus/CAE is the Complete Abaqus Environment that provides a simple, consistent interface for creating Abaqus models, interactively submitting and monitoring Abaqus jobs, and evaluating results from Abaqus simulations.

Abaqus/CAE is divided into modules, where each module defines a logical aspect of the modeling process; for example, defining the geometry, defining material properties, and generating a mesh. As you move from module to module, you build up the model. When the model is complete, Abaqus/CAE generates an input file that you submit to the Abaqus analysis product. Abaqus/Standard or Abaqus/Explicit reads the input file generated by Abaqus/CAE, performs the analysis, sends information to Abaqus/CAE to allow you to monitor the progress of the job, and generates an output database. Finally, you use the Visualization module to read the output database and view the results of your analysis.

In this section:

- *Starting Abaqus/CAE*
- *Components of the main window*
- *What is a module?*
- *What is the Model Tree?*

Starting Abaqus/CAE

To start Abaqus/CAE, you enter the command

```
abaqus cae
```

at your operating system prompt, where *abaqus* is the command used to run Abaqus. This command may be different on your system.

When Abaqus/CAE begins, the **Start Session** dialog box appears. The following session startup options are available:

- **Create Model Database: With Standard/Explicit Model** allows you to begin a new Abaqus/Standard or Abaqus/Explicit analysis.
- **Create Model Database: With Electromagnetic Model** allows you to begin a new electromagnetic analysis.
- **Open Database** allows you to open a previously saved model or output database file.
- **Run Script** allows you to run a file containing Abaqus/CAE commands.
- **Start Tutorial** allows you to begin an introductory tutorial from the online documentation.
- **Recent Files** allows you to open one of the five most recently opened model or output database files.

Components of the main window

You interact with Abaqus/CAE through the main window. *Figure 1* shows the components that appear in the main window.

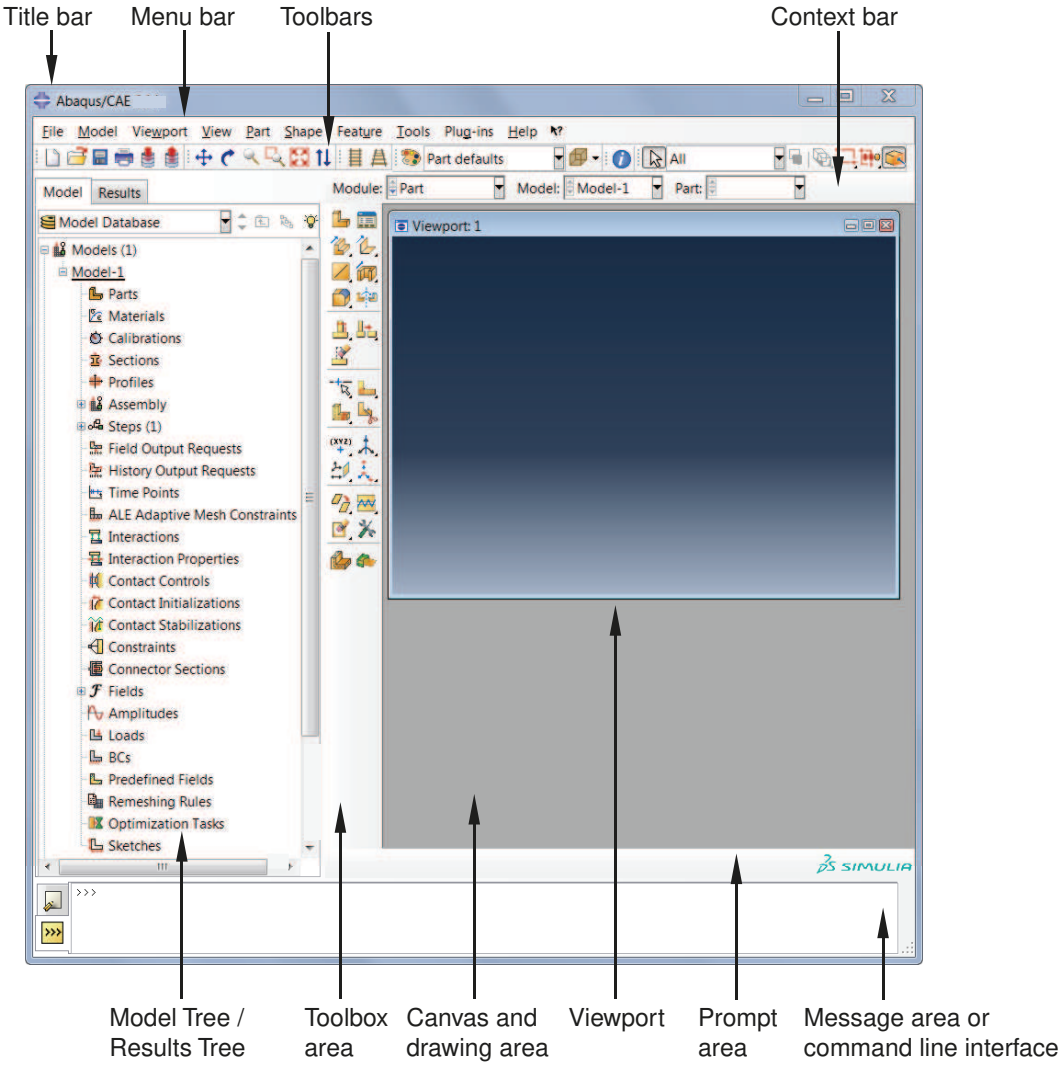


Figure 1: Components of the main window.

The components are:

Title bar

The title bar indicates the release of Abaqus/CAE you are running and the name of the current model database.

Menu bar

The menu bar contains all the available menus; the menus give access to all the functionality in the product. Different menus appear in the menu bar depending on which module you selected from the context bar. For more information, see *Components of the main menu bar*.

Toolbars

The toolbars provide quick access to items that are also available in the menus. For more information, see *Components of the toolbars*.

Context bar

Abaqus/CAE is divided into a set of modules, where each module allows you to work on one aspect of your model; the **Module** list in the context bar allows you to move between these modules. Other items in the context bar are a function of the module in which you are working; for example, the context bar allows you to retrieve an existing part while creating the geometry of the model. For more information, see *The context bar*.

Model Tree

The Model Tree provides you with a graphical overview of your model and the objects that it contains, such as parts, materials, steps, loads, and output requests. In addition, the Model Tree provides a convenient, centralized tool for moving between modules and for managing objects. If your model database contains more than one model, you can use the Model Tree to move between models. When you become familiar with the Model Tree, you will find that you can quickly perform most of the actions that are found in the main menu bar, the module toolboxes, and the various managers. For more information, see *Working with the Model Tree and the Results Tree*.

Results Tree

The Results Tree provides you with a graphical overview of your output databases and other session-specific data such as *X–Y* plots. If you have more than one output database open in your session, you can use the Results Tree to move between output databases. When you become familiar with the Results Tree, you will find that you can quickly perform most of the actions in the Visualization module that are found in the main menu bar and the toolbox. For more information, see *The Results Tree*.

Toolbox area

When you enter a module, the toolbox area displays tools in the toolbox that are appropriate for that module. The toolbox allows quick access to many of the module functions that are also available from the menu bar. For more information, see *Understanding and using toolboxes and toolbars*.

Canvas and drawing area

The canvas can be thought of as an infinite screen or bulletin board on which you post viewports; for more information, see *Managing viewports on the canvas*. The drawing area is the visible portion of the canvas.

Viewport

Viewports are windows on the canvas in which Abaqus/CAE displays your model. For more information, see *Managing viewports on the canvas*.

Prompt area

The prompt area displays instructions for you to follow during a procedure; for example, it asks you to select the geometry as you create a set. For more information, see *Using the prompt area during procedures*.

Message area

Abaqus/CAE prints status information and warnings in the message area. To resize the message area, drag the top edge; to see information that has scrolled out of the message area, use the scroll bar on the right side. The message area is displayed by default, but it uses the same space occupied by the command line interface. If you have recently used the command line interface, you must click the  tab in the bottom left corner of the main window to activate the message area.



Note:

If new messages are added while the command line interface is active, Abaqus/CAE changes the background color surrounding the message area icon to red. When you display the message area, the background reverts to its normal color.

Command line interface

You can use the command line interface to type Python commands and evaluate mathematical expressions using the Python interpreter that is built into Abaqus/CAE. The interface includes primary (>>>) and secondary (`...`) prompts to indicate when you must indent commands to comply with Python syntax.

The command line interface is hidden by default, but it uses the same space occupied by the message area. Click the  tab in the bottom left corner of the main window to switch from the message area to the command line interface. Click the  tab to return to the message area.

What is a module?

As mentioned earlier, Abaqus/CAE is divided into functional units called modules. Each module contains only those tools that are relevant to a specific portion of the modeling task. For example, the Mesh module contains only the tools needed to create finite element meshes, while the Job module contains only the tools used to create, edit, submit, and monitor analysis jobs.

You select a module from the **Module** list in the context bar, as shown in *Figure 1*.



Figure 1: Selecting a module.

The order of the modules in the menu corresponds to a logical sequence you may follow to create a model. In many circumstances you must follow this natural progression to complete a modeling task; for example, you must create parts before you create an assembly. Although the order of the modules follows a logical sequence, Abaqus/CAE allows you to select any module at any time, regardless of the state of your model. However, certain obvious restrictions apply; for example, you cannot assign section properties, such as cross-sectional dimensions of an I-beam, to geometry that has not yet been created.

A completed model contains everything that Abaqus needs to start the analysis. Abaqus/CAE uses a model database to store your models. When you start Abaqus/CAE, the **Start Session** dialog box allows you to create a new, empty model database in memory. After you start Abaqus/CAE, you can save your model database to a disk by selecting **File->Save** from the main menu bar; to retrieve a model database from a disk, select **File->Open**.

The following list of the modules available within Abaqus/CAE briefly describes the modeling tasks you can perform in each module. The order of the modules in the list corresponds to the order of the modules in the context bar's **Module** list (see *Figure 1*):

Part

The Part module allows you to create individual parts by sketching their geometry directly in Abaqus/CAE or by importing their geometry from other geometric modeling programs. For more information, see *The Part module*.

Property

A section definition contains information about the properties of a part or a region of a part, such as a region's associated material definition and cross-sectional geometry. In the Property module you create section and material definitions and assign them to regions of parts. For more information, see *The Property module*.

Assembly

When you create a part, it exists in its own coordinate system, independent of other parts in the model. You use the Assembly module to create instances of your parts as well as instances of other models and to position the instances relative to each other in a global coordinate system, thus creating an assembly. An Abaqus model contains only one assembly. For more information, see *The Assembly module*.

Step

You use the Step module to create and configure analysis steps and associated output requests. The step sequence provides a convenient way to capture changes in a model (such as loading and boundary condition changes); output requests can vary as necessary between steps. For more information, see *The Step module*.

Interaction

In the Interaction module you specify mechanical and thermal interactions between regions of a model or between a region of a model and its surroundings. An example of an interaction is contact between two surfaces. Other interactions that may be defined include constraints, such as tie, equation, and rigid body constraints. Abaqus/CAE does not recognize mechanical contact between part instances or regions of an assembly unless that contact is specified in the Interaction module; the mere physical proximity of two surfaces in an assembly is not sufficient to indicate any type of interaction between the surfaces. Interactions are step-dependent objects, which means that you must specify the analysis steps in which they are active. For more information, see *The Interaction module*.

Load

The Load module allows you to specify loads, boundary conditions, and predefined fields. Loads and boundary conditions are step-dependent objects, which means that you must specify the analysis steps in which they are active; some predefined fields are step-dependent, while others are applied only at the beginning of the analysis. For more information, see *The Load module*.

Mesh

The Mesh module contains tools that allow you to generate a finite element mesh on an assembly created within Abaqus/CAE. Various levels of automation and control are available so that you can create a mesh that meets the needs of your analysis. For more information, see *The Mesh module*.

Optimization

The Optimization module allows you to create an optimization task that can be used to optimize the topology of your model given a set of objectives and constraints. For more information, see *The Optimization module*.

Job

Once you have finished all of the tasks involved in defining a model, you use the Job module to analyze your model. The Job module allows you to interactively submit a job for analysis and monitor its progress. Multiple models and runs may be submitted and monitored simultaneously. For more information, see *The Job module*.

Visualization

The Visualization module provides graphical display of finite element models and results. It obtains model and result information from the output database; you can control what information is written to the output database by modifying output requests in the Step module. For more information, see *Viewing results*.

Sketch

Sketches are two-dimensional profiles that are used to help form the geometry defining an Abaqus/CAE native part. You use the Sketch module to create a sketch that defines a planar part, a beam, or a partition or to create a sketch that might be extruded, swept, or revolved to form a three-dimensional part. For more information, see *The Sketch module*.

The contents of the main window change as you move between modules. Selecting a module from the **Module** list on the context bar causes the context bar, module toolbox, and menu bar to change to reflect the functionality of the current module.

Each module is discussed in more detail in the tutorial examples presented in this guide.

What is the Model Tree?

The Model Tree provides a visual description of the hierarchy of items in a model. It is located in the left side of the main window underneath the **Model** tab. *Figure 1* shows a typical Model Tree.

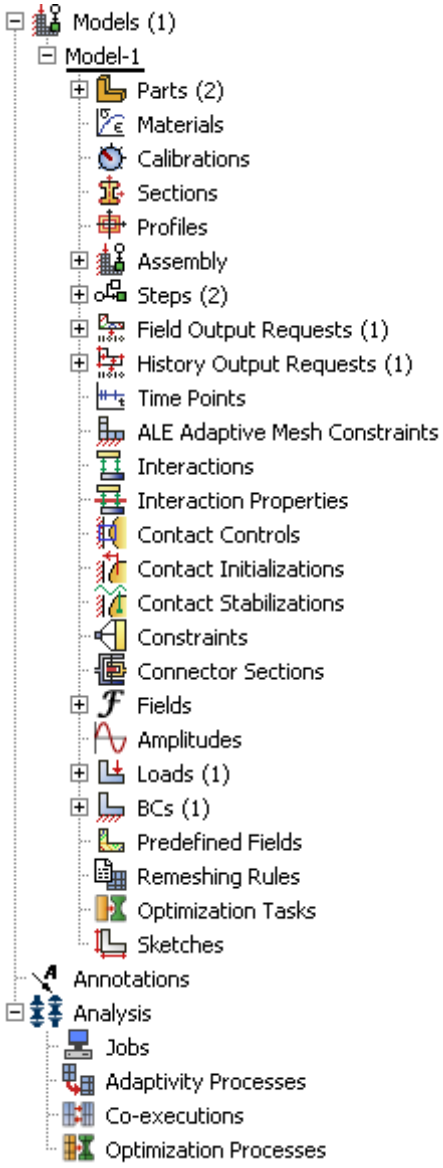
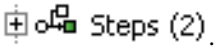


Figure 1: Model Tree.

Items in the Model Tree are represented by small icons; for example, the **Steps** icon, . In addition, parentheses next to an item indicate that the item is a container, and the number in the parentheses indicates the number of items in the container. You can click on the "+" and "−" signs in the Model Tree to expand and collapse a container. The right and left arrow keys perform the same operation.

The arrangement of the containers and items in the Model Tree reflects the order in which you are expected to create your model. As noted earlier, a similar logic governs the order of modules in the module menu—you create parts before you create the assembly, and you create steps before you create loads. This arrangement is fixed—you cannot move items in the Model Tree.

The Model Tree provides most of the functionality of the main menu bar and the module managers. For example, if you double-click on the **Parts** container, you can create a new part (the equivalent of selecting **Part->Create** from the main menu bar).

The instructions for the examples discussed in this guide will focus on using the Model Tree to access the functionality of Abaqus/CAE. Menu bar actions will be considered only when necessary (e.g., when creating a finite element mesh or postprocessing results).

The Results Tree uses the same space occupied by the Model Tree. Click the **Results** tab in the left side of the main window to switch from the Model Tree to the Results Tree. The Results Tree provides access to session-specific features (i.e., functionality available in only the Visualization module). The Results Tree will be introduced during the course of the postprocessing exercises contained in this guide.

Example: creating a model of an overhead hoist

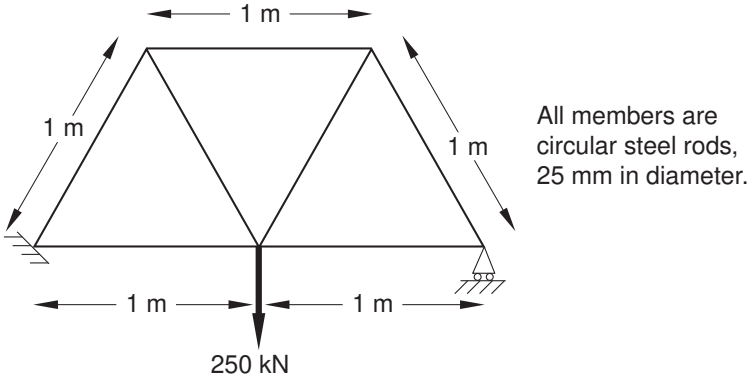
This example of an overhead hoist leads you through the Abaqus/CAE modeling process by using the Model Tree and showing you the basic steps used to create and analyze a simple model.

In this section:

- *Overhead hoist analysis*
- *Units*
- *Creating a part*
- *Creating a material*
- *Defining and assigning section properties*
- *Defining the assembly*
- *Configuring your analysis*
- *Applying boundary conditions and loads to the model*
- *Meshing the model*
- *Creating an analysis job*
- *Checking the model*
- *Running the analysis*
- *Postprocessing*
- *Rerunning the analysis using Abaqus/Explicit*
- *Postprocessing the dynamic analysis results*

Overhead hoist analysis

The hoist is a simple, pin-jointed truss model that is constrained at the left end and mounted on rollers at the right end. The members can rotate freely at the joints. The frame is prevented from moving out of plane. A simulation is first performed in Abaqus/Standard to determine the structure's static deflection and the peak stress in its members when a 250 kN load is applied as shown in *Figure 1*. The simulation is performed a second time in Abaqus/Explicit under the assumption that the load is applied suddenly to study the dynamic response of the frame.



Material properties

General properties:

$\rho = 7800\ \text{kg/m}^3$

Elastic properties:

$E = 200 \times 10^9\ \text{Pa}$

$\nu = 0.3$

Figure 1: Schematic of an overhead hoist.

For the overhead hoist example, you will perform the following tasks:

- Sketch the two-dimensional geometry and create a part representing the frame.
- Define the material properties and section properties of the frame.
- Assemble the model.
- Configure the analysis procedure and output requests.
- Apply loads and boundary conditions to the frame.
- Mesh the frame.
- Create a job and submit it for analysis.
- View the results of the analysis.

Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Overhead hoist frame*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Overhead hoist frame**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Units

Before starting to define this or any model, you need to decide which system of units you will use. Abaqus has no built-in system of units.
Do not include unit names or labels when entering data in Abaqus. All input data must be specified in consistent units. Some common systems of consistent units are shown in *Table 1*.

Table 1: Consistent units.

Quantity	SI	SI (mm)	US Unit (ft)	US Unit (inch)
Length	m	mm	ft	in
Force	N	N	lbf	lbf
Mass	kg	tonne (10^3 kg)	slug	lbf s^2/in
Time	s	s	s	s
Stress	Pa (N/m^2)	MPa (N/mm^2)	lbf/ft^2	psi (lbf/in^2)
Energy	J	mJ (10^{-3} J)	ft lbf	in lbf
Density	kg/m^3	$tonne/mm^3$	$slug/ft^3$	lbf s^2/in^4

The SI system of units is used throughout this guide. Users working in the systems labeled “US Unit” should be careful with the units of density; often the densities given in handbooks of material properties are multiplied by the acceleration due to gravity.

Creating a part

Parts define the geometry of the individual components of your model and, therefore, are the building blocks of an Abaqus/CAE model. You can create parts that are native to Abaqus/CAE, or you can import parts created by other applications either as a geometric representation or as a finite element mesh.

You will start the overhead hoist problem by creating a two-dimensional, deformable wire part. You do this by sketching the geometry of the frame. Abaqus/CAE automatically enters the Sketcher when you create a part.

Abaqus/CAE often displays a short message in the prompt area indicating what you should do next, as shown in *Figure 1*.

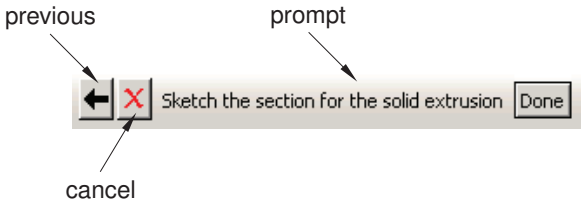


Figure 1: Messages and instructions are displayed in the prompt area.

Click the **Cancel** button to cancel the current task. Click the **Previous** button to cancel the current step in the task and return to the previous step.

1. If you did not already start Abaqus/CAE, type `abaqus cae`, where `abaqus` is the command used to run Abaqus.
2. From the **Create Model Database** options in the **Start Session** dialog box that appears, select **With Standard/Explicit Model**.

   Abaqus/CAE enters the Part module. The Model Tree appears in the left side of the main window (underneath the **Model** tab). Between the Model Tree and the canvas is the Part module toolbox. A toolbox contains a set of icons that allow expert users to bypass the menus in the main menu bar. For many tools, as you select an item from the main menu bar or the Model Tree, the corresponding tool is highlighted in the module toolbox so you can learn its location.
3. In the Model Tree, double-click the **Parts** container to create a new part.

   The **Create Part** dialog box appears. Abaqus/CAE also displays text in the prompt area near the bottom of the window to guide you through the procedure.

   You use the **Create Part** dialog box to name the part; to choose its modeling space, type, and base feature; and to set the approximate size. You can edit and rename a part after you create it; you can also change its modeling space and type but not its base feature.
4. Name the part `Frame`. Choose a two-dimensional planar deformable body and a wire base feature.
5. In the **Approximate size** text field, type `4.0`.

   The value entered in the **Approximate size** text field at the bottom of the dialog box sets the approximate size of the new part. The size that you enter is used by Abaqus/CAE to calculate the size of the Sketcher sheet and the spacing of its grid. You should choose this value to be on the order of the largest dimension of your finished part. Recall that Abaqus/CAE does not use specific units, but the units must be consistent throughout the model. In this model SI units will be used.
6. Click **Continue** to exit the **Create Part** dialog box.

   Abaqus/CAE automatically enters the Sketcher. The Sketcher toolbox appears in the left side of the main window, and the Sketcher grid appears in the viewport. The Sketcher contains a set of basic tools that allow you to sketch the two-dimensional profile of your part. Abaqus/CAE enters the Sketcher whenever you create or edit a part. To finish using any tool, click mouse button 2 in the viewport or select a new tool.

Tip: Like all tools in Abaqus/CAE, if you simply position the cursor over a tool in the Sketcher toolbox for a short time, a small window appears that gives a brief description of the tool. When you select a tool, a white background appears on it.

The following aspects of the Sketcher help you sketch the desired geometry:

- The Sketcher grid helps you position the cursor and align objects in the viewport.
- Dashed lines indicate the *X*- and *Y*-axes of the sketch and intersect at the origin of the sketch.
- A triad in the lower-left corner of the viewport indicates the relationship between the sketch plane and the orientation of the part.
- When you select a sketching tool, Abaqus/CAE displays the *X*- and *Y*-coordinates of the cursor in the upper-left corner of the viewport.

7. You will first sketch a rough approximation of the frame and later use constraints and dimensions to refine the sketch. Begin by using the **Create Lines: Rectangle** tool located in the upper-right region of the Sketcher toolbox to sketch an arbitrary rectangle. Select any two points as the opposite corners of the rectangle.

Click mouse button 2 anywhere in the viewport to exit the rectangle tool.

Note:

If you make a mistake while using the Sketcher, you can undo your last action using the **Undo** tool or delete individual entities of your sketch using the **Delete** tool .

8. The Sketcher automatically adds constraints to the sketch as indicated in *Figure 2* (in this case, the four corners of the rectangle are assigned perpendicular constraints and one edge is designated as horizontal).

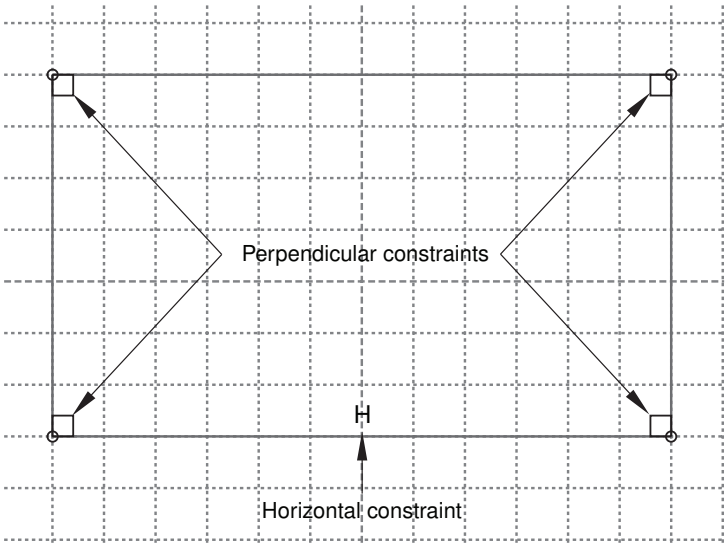


Figure 2: Constraints indicated in the Sketcher.

To proceed, the perpendicular constraints must be deleted. In the Sketcher toolbox, select the **Delete** tool and then do the following:

a. In the prompt area, select **Constraints** as the scope of the operation.

b. Using **[Shift] + Click**, select the four perpendicular constraints.

c. Click **Done** in the prompt area.

9. You will now add additional constraints and dimensions to refine the sketch. Constraints and dimensions allow you to control your sketch geometry and add precision. For more information on constraints and dimensions, see *Controlling sketch geometry*.

a. Use the **Add Constraint** tool to constrain the top and bottom edges so they remain parallel to each other:

1. In the **Add Constraint** dialog box, select **Parallel**.
2. In the viewport, select the top and bottom edges of the sketch (using **[Shift] + Click**).
3. Click **Done** in the prompt area.

b. Use the **Add Dimension** tool to dimension the top and bottom edges of the rectangle. The top edge should have a horizontal dimension of `1` m and the bottom edge a horizontal dimension of `2` m. When dimensioning each edge, simply select the line, click mouse button 1 to position the dimension text, and then enter the new dimension in the prompt area. Selecting the line rather than its endpoints constrains the length of the line regardless of its orientation in space (in effect, defining an oblique dimension).

Reset the view as needed using the **Auto-Fit View** tool in the View Manipulation toolbar to see the updated sketch.

c. Dimension the left and right edges so they each have an oblique dimension of `1` m.

The sketch in its current state is shown in *Figure 3*. In this figure the default grid spacing has been doubled. For information on using the **Sketcher Options** tool to modify the Sketcher display, see *Customizing the Sketcher*.

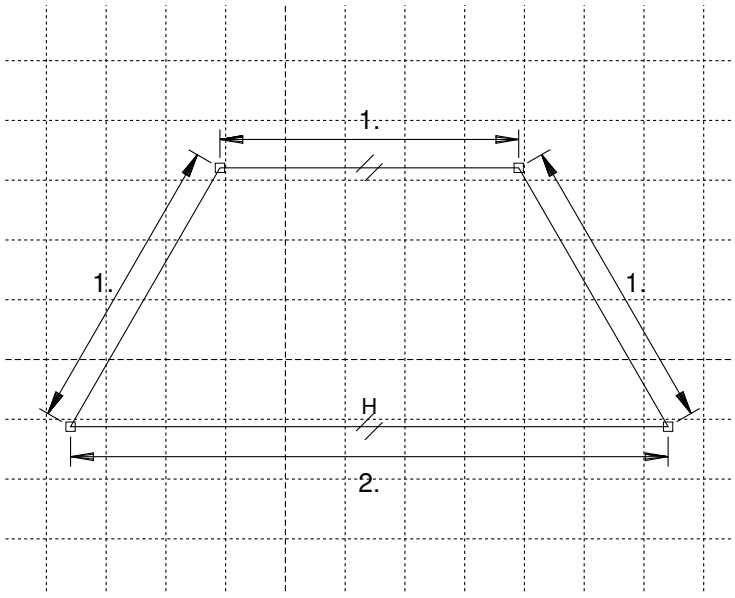


Figure 3: Rough sketch of frame (with grid spacing doubled).

10. Now sketch the interior edges of the frame.

a. Using the **Create Lines: Connected** tool  located in the upper-right corner of the Sketcher toolbox, create two lines as follows:

1. Start the first line at the upper-left corner of the sketch and extend it to any point that snaps onto the bottom edge (its horizontal location is arbitrary).

2. Continue the second line to the upper-right corner of the sketch.

3. Click mouse button 2 anywhere in the viewport to exit the connected lines tool.

b. Using the **Split** tool , split the bottom edge at the point where it intersects the two lines created above:

1. Note the small black triangles at the base of some of the toolbox icons. These triangles indicate the presence of hidden icons that can be revealed. Click and hold the **Auto-Trim** tool  located on the middle-right of the Sketcher toolbox until additional icons appear.

2. From the set of additional icons, click the **Split** tool .

The split tool appears in the Sketcher toolbox with a white background indicating that you selected it.

3. Select the bottom edge as the first entity to define the split point.

4. Select either of the two interior lines as the second entity (a red circle will appear around the split point).

5. Click mouse button 2 to indicate that you have finished using the split tool.

c. Use the **Add Constraint** tool  to constrain the two segments of the bottom edge so they are of equal length:

1. In the **Add Constraint** dialog box, select **Equal length**.

2. In the viewport, select the two segments of the bottom edge.

3. Click **Done** in the prompt area.

11. The final sketch is shown in *Figure 4*.

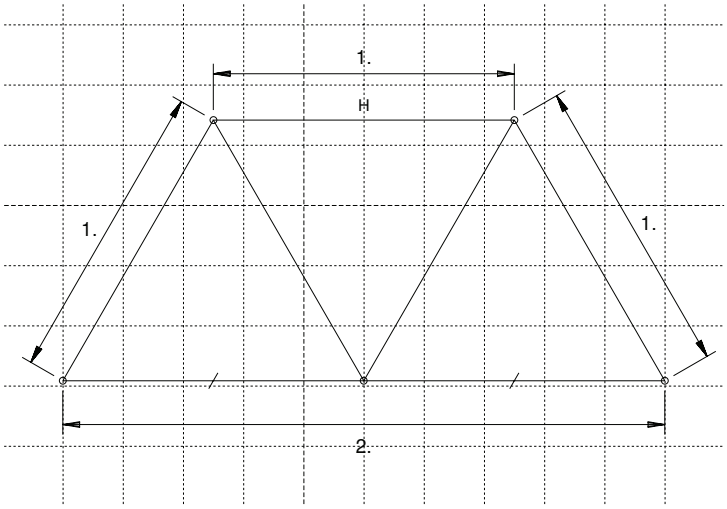


Figure 4: Frame geometry sketch.

12. From the prompt area (near the bottom of the main window), click **Done** to exit the Sketcher.

Note:

If you do not see the **Done** button in the prompt area, continue to click mouse button 2 in the viewport until it appears.

13. Before you continue, save your model in a model database file.

a. From the main menu bar, select **File->Save**. The **Save Model Database As** dialog box appears.

b. Type a name for the new model database in the **File Name** field, and click **OK**. You do not need to include the file extension; Abaqus/CAE automatically appends `.cae` to the file name.

Abaqus/CAE stores the model database in a new file and returns to the Part module. The path and name of your model database appear in the main window title bar.

You should always save your model database at regular intervals (for example, each time you switch modules); Abaqus/CAE does not save your model database automatically.

Creating a material

In this problem all the members of the frame are made of steel and assumed to be linear elastic with Young's modulus of 200 GPa and Poisson's ratio of 0.3. Thus, you will create a single linear elastic material with these properties.

1. In the Model Tree, double-click the **Materials** container to create a new material.

   Abaqus/CAE switches to the Property module, and the **Edit Material** dialog box appears.

2. Name the material `Steel`.
3. Use the menu bar under the browser area of the material editor to reveal menus containing all the available material options. Some of the menu items contain submenus; for example, *Figure 1* shows the options available under the **Mechanical->Elasticity** menu item. When you select a material option, the appropriate data entry form appears below the menu.

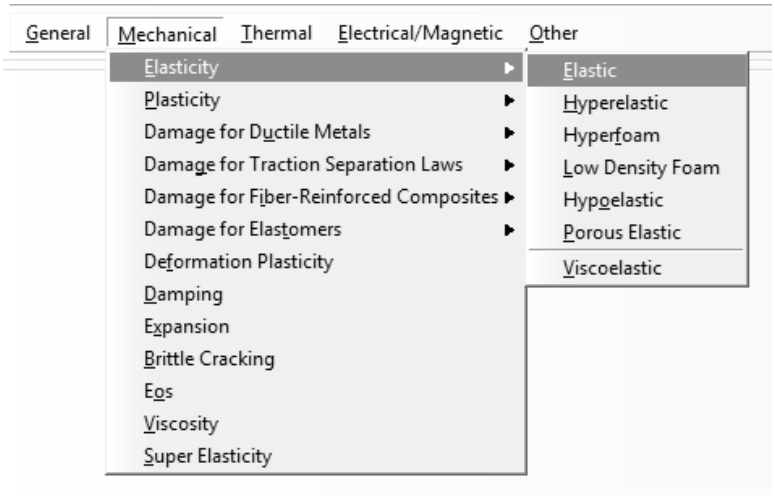


Figure 1: Submenus available under the Mechanical->Elasticity menu.

4. From the material editor's menu bar, select **Mechanical->Elasticity->Elastic**.

   Abaqus/CAE displays the **Elastic** data form.

5. Type a value of `200.0E9` for Young's modulus and a value of `0.3` for Poisson's ratio in the respective fields. Use **[Tab]** or move the cursor to a new cell and click to move between cells.
6. Click **OK** to exit the material editor.

Defining and assigning section properties

You define the properties of a part through sections.
After you create a section, you can use one of the following two methods to assign the section to the part in the current viewport:

- You can simply select the region from the part and assign the section to the selected region.
- You can use the Set toolset to create a homogeneous set containing the region and assign the section to the set.

For the frame model you will create a single truss section that you will assign to the frame by selecting the frame from the viewport. The section will refer to the material `Steel` that you just created as well as define the cross-sectional area of the frame members.

In this section:

- *Defining a truss section*
- *Assigning the section to the frame*

Defining a truss section

A truss section definition requires only a material reference and the cross-sectional area. Remember that the frame members are circular bars that are 0.025 m in diameter. Thus, their cross-sectional area is 49.0874×10^{-5} m^2.



Tip: You can use the command line interface (CLI) in Abaqus/CAE as a simple calculator. For example, to compute the cross-sectional area of the frame members, click the  tab in the bottom left corner of the Abaqus/CAE window to activate the CLI, type `pi*0.025**2/4.0` after the command prompt, and press **[Enter]**. The value of the cross-sectional area is printed in the CLI.

1. In the Model Tree, double-click the **Sections** container to create a section.

   The **Create Section** dialog box appears.

2. In the **Create Section** dialog box:
   a. Name the section `FrameSection`.
   b. In the **Category** list, select **Beam**.
   c. In the **Type** list, select **Truss**.
   d. Click **Continue**.

   The **Edit Section** dialog box appears.

3. In the **Edit Section** dialog box:
   a. Accept the default selection of `Steel` for the **Material** associated with the section. If you had defined other materials, you could click the arrow next to the **Material** text box to see a list of available materials and to select the material of your choice.
   b. In the **Cross-sectional area** field, enter `pi*0.025**2/4.0`.

      Tip: If a field in a dialog box is expecting a floating point number, you can enter an arithmetic expression instead. The expression is evaluated by the Python interpreter that is built into Abaqus/CAE. The arithmetic expression will be replaced by its value.

   c. Click **OK**.

Assigning the section to the frame

The section `FrameSection` must be assigned to the frame.

1. In the Model Tree, expand the branch for the part named **Frame** by clicking the "+" symbol to expand the **Parts** container and then clicking the "+" symbol to expand the **Frame** item.
2. Double-click **Section Assignments** in the list of part attributes that appears.

   Abaqus/CAE displays prompts in the prompt area to guide you through the procedure.
3. Select the entire part as the region to which the section will be applied.
   a. Click and hold mouse button 1 at the upper left-hand corner of the viewport.
   b. Drag the mouse to create a box around the truss.
   c. Release mouse button 1.

   Abaqus/CAE highlights the entire frame.
4. In the prompt area, enter `all` as the name of the set that will be created.

   This set will contain the selected regions. The advantage of creating a set is that it can be used for other purposes. If you prefer to not create a set, toggle off **Create set** in the prompt area.
5. Click mouse button 2 in the viewport or click **Done** in the prompt area to accept the selected geometry.

   The **Edit Section Assignment** dialog box appears containing a list of existing sections.
6. Accept the default selection of `FrameSection`, and click **OK**.

   Abaqus/CAE assigns the truss section to the frame, colors the entire frame aqua to indicate that the region has a section assignment, and closes the **Edit Section Assignment** dialog box.

Defining the assembly

Each part that you create is oriented in its own coordinate system and is independent of the other parts in the model. Although a model may contain many parts, it contains only one assembly. You define the geometry of the assembly by creating instances of a part and then positioning the instances relative to each other in a global coordinate system. An instance can be classified as either independent or dependent. Independent part instances are meshed individually, while the mesh of a dependent part instance is associated with the mesh of the original part. For further details, see *Working with part instances*. By default, part instances are dependent.

For this problem you will create a single instance of your overhead hoist. Abaqus/CAE positions the instance so that the origin of the sketch that defined the frame overlays the origin of the assembly's default coordinate system.

1. In the Model Tree, expand the **Assembly** container and double-click **Instances** in the list that appears.
   Abaqus/CAE switches to the Assembly module, and the **Create Instance** dialog box appears.
2. In the dialog box, select **Parts** to choose parts from the current model.
3. Select **Frame** and click **OK**.

Abaqus/CAE creates an instance of the overhead hoist. In this example the single instance of the frame defines the assembly. The frame is displayed in the 1–2 plane of the global coordinate system (a right-handed, rectangular Cartesian system). A triad in the lower-left corner of the viewport indicates the orientation of the model with respect to the view. A second triad in the viewport indicates the origin and orientation of the global coordinate system (*X*-, *Y*-, and *Z*-axes). The global 1-axis is the horizontal axis of the hoist, the global 2-axis is the vertical axis, and the global 3-axis is normal to the plane of the framework. For two-dimensional problems such as this one Abaqus requires that the model lie in a plane parallel to the global 1–2 plane.

Configuring your analysis

Now that you have created your assembly, you can configure your analysis.
In this simulation we are interested in the static response of the overhead hoist to a 10 kN load applied at the midspan, with the left-hand end fully constrained and a roller constraint on the right-hand end (see *Figure 1*). This is a single event, so only a single analysis step is needed for the simulation. Thus, the model will consist of two steps overall:

- An initial step, in which you will apply boundary conditions that constrain the ends of the frame.
- An analysis step, in which you will apply a concentrated load at the midspan of the frame.

Abaqus/CAE generates the initial step automatically, but you must create the analysis step yourself. You may also request output for any steps in the analysis.

There are two kinds of analysis 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. Only general analysis steps are available in Abaqus/Explicit. For this simulation you will define a static linear perturbation step. Perturbation procedures are discussed further in *Multiple Step Analysis*.

In this section:

- *Creating an analysis step*
- *Requesting data output*

Creating an analysis step

Create a static, linear perturbation step that follows the initial step of the analysis.

1. In the Model Tree, double-click the **Steps** container to create a step.

   Abaqus/CAE switches to the Step module, and the **Create Step** dialog box appears. A list of all the general procedures and a default step name of `Step-1` is provided.

2. Change the step name to `Apply load`.
3. Select **Linear perturbation** as the **Procedure type**.
4. From the list of available linear perturbation procedures in the **Create Step** dialog box, select **Static, Linear perturbation** and click **Continue**.

   The **Edit Step** dialog box appears with the default settings for a static linear perturbation step.

5. The **Basic** tab is selected by default. In the **Description** field, type `10 kN central load`.
6. Click the **Other** tab to see its contents; you can accept the default values provided for the step.
7. Click **OK** to create the step and to exit the **Edit Step** dialog box.

Requesting data output

Finite element analyses can create very large amounts of output. Abaqus allows you to control and manage this output so that only data required to interpret the results of your simulation are produced. Four types of output are available from an Abaqus analysis:

- Results stored in a neutral binary file used by Abaqus/CAE for postprocessing. This file is called the Abaqus output database file and has the extension `.odb`.
- Printed tables of results, written to the Abaqus data (`.dat`) file. Output to the data file is available only in Abaqus/Standard.
- Restart data used to continue the analysis, written to the Abaqus restart (`.res`) file.
- Results stored in binary files for subsequent postprocessing with third-party software, written to the Abaqus results (`.fil`) file.

You will use only the first of these in the overhead hoist simulation.

By default, Abaqus/CAE writes the results of the analysis to the output database (`.odb`) file. When you create a step, Abaqus/CAE generates a default output request for the step. A list of the preselected variables written by default to the output database is given in the *Abaqus Output Guide*. You do not need to do anything to accept these defaults. You use the **Field Output Requests Manager** to request output of variables that should be written at relatively low frequencies to the output database from the entire model or from a large portion of the model. You use the **History Output Requests Manager** to request output of variables that should be written to the output database at a high frequency from a small portion of the model; for example, the displacement of a single node.

For this example you will examine the output requests to the `.odb` file and accept the default configuration.

1. In the Model Tree, click mouse button 3 on the **Field Output Requests** container and select **Manager** from the menu that appears.

   Abaqus/CAE displays the **Field Output Requests Manager**. This manager displays the status of field output requests in a table format. The left side of the table has an alphabetical list of existing output requests. The top of the table lists the names of all the steps in the analysis in the order of execution. Each cell of the table displays the status of each output request in each step.

   You can use the **Field Output Requests Manager** to do the following:

   - Select the variables that Abaqus will write to the output database.
   - Select the section points for which Abaqus will generate output data.
   - Select the region of the model for which Abaqus will generate output data.
   - Change the frequency at which Abaqus will write data to the output database.

2. Review the default output request that Abaqus/CAE generates for the **Static, Linear perturbation** step you created and named `Apply load`.

   Select the cell in the table labeled `Created` if it is not already selected. The following information related to the cell is shown in the legend at the bottom of the manager:

   - The type of analysis procedure carried out in the step in that column.
   - The list of output request variables.
   - The output request status.

3. On the right side of the **Field Output Requests Manager**, click **Edit** to view more detailed information about the output request.

The field output editor appears. In the **Output Variables** region of this dialog box, there is a text box that lists all variables that will be output. If you change an output request, you can always return to the default settings by choosing **Preselected defaults** above the text box.

4. Click the arrows next to each output variable category to see exactly which variables will be output. The boxes next to each category title allow you to see at a glance whether all variables in that category will be output. A black check mark indicates that all variables are output, while a gray check mark indicates that only some variables will be output.

   Based on the selections shown at the bottom of the dialog box, data will be generated at every default section point in the model and will be written to the output database after every increment during the analysis.

5. Click **Cancel** to close the field output editor, since you do not wish to make any changes to the default output requests.
6. Click **Dismiss** to close the **Field Output Requests Manager**.

   Note:

   What is the difference between the **Dismiss** and **Cancel** buttons? Dismiss buttons appear in dialog boxes that contain data that you cannot modify. For example, the **Field Output Requests Manager** allows you to view output requests, but you must use the field output editor to modify those requests. Clicking the **Dismiss** button simply closes the **Field Output Requests Manager**. Conversely, **Cancel** buttons appear in dialog boxes that allow you to make changes. Clicking **Cancel** closes the dialog box without saving your changes.

7. Review the history output requests in a similar manner by right-clicking the **History Output Requests** container in the Model Tree and opening the history output editor.

Applying boundary conditions and loads to the model

Prescribed conditions, such as loads and boundary conditions, are step dependent, which means that you must specify the step or steps in which they become active.
Now that you have defined the steps in the analysis, you can define prescribed conditions.

In this section:

- *Applying boundary conditions to the frame*
- *Applying a load to the frame*

Applying boundary conditions to the frame

In structural analyses, boundary conditions are applied to those regions of the model where the displacements and/or rotations are known. Such regions may be constrained to remain fixed (have zero displacement and/or rotation) during the simulation or may have specified, nonzero displacements and/or rotations.

In this model the bottom-left portion of the frame is constrained completely and, thus, cannot move in any direction. The bottom-right portion of the frame, however, is fixed in the vertical direction but is free to move in the horizontal direction. The directions in which motion is possible are called *degrees of freedom (dof)*.

The labeling convention used for the displacement and rotational degrees of freedom in Abaqus is shown in *Figure 1*.

1 Translation in the 1-direction (U1).

2 Translation in the 2-direction (U2).

3 Translation in the 3-direction (U3).

4 Rotation about the 1-direction (UR1).

5 Rotation about the 2-direction (UR2).

6 Rotation about the 3-direction (UR3).

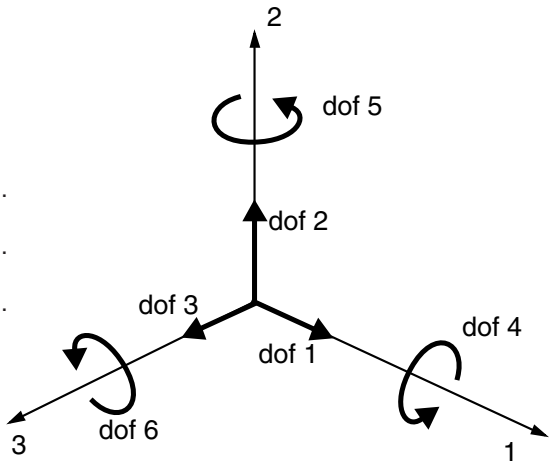


Figure 1: Displacement and rotational degrees of freedom.

In this example all the constraints are in the global 1- or 2-directions. In many cases constraints are required in directions that are not aligned with the global directions. In such cases you can define a local coordinate system for boundary condition application. The skew plate example in *Using Shell Elements* demonstrates how to do this.

1. In the Model Tree, double-click the **BCs** container.

   Abaqus/CAE switches to the Load module, and the **Create Boundary Condition** dialog box appears.

2. In the **Create Boundary Condition** dialog box:
   a. Name the boundary condition `Fixed`.
   b. From the list of steps, select `Initial` as the step in which the boundary condition will be activated. All the mechanical boundary conditions specified in the `Initial` step must have zero magnitudes. This condition is enforced automatically by Abaqus/CAE.
   c. In the **Category** list, accept **Mechanical** as the default category selection.
   d. In the **Types for Selected Step** list, select **Displacement/Rotation**, and click **Continue**.

   Abaqus/CAE displays prompts in the prompt area to guide you through the procedure. For example, you are asked to select the region to which the boundary condition will be applied.

   To apply a prescribed condition to a region, you can either select the region directly in the viewport or apply the condition to an existing set (a set is a named region of a model). Sets are a convenient tool that can be used to manage large complicated models. In this simple model you will not make use of sets.

3. In the viewport, select the vertex at the bottom-left corner of the frame as the region to which the boundary condition will be applied. Name the associated set `left`.
4. Click mouse button 2 in the viewport or click **Done** in the prompt area to indicate that you have finished selecting regions.

The **Edit Boundary Condition** dialog box appears. When you are defining a boundary condition in the initial step, all available degrees of freedom are unconstrained by default.

5. In the dialog box:
   a. Toggle on **U1** and **U2** since all translational degrees of freedom need to be constrained.
   b. Click **OK** to create the boundary condition and to close the dialog box.

      Abaqus/CAE displays two arrowheads at the vertex to indicate the constrained degrees of freedom.

6. Repeat the above procedure to constrain degree of freedom **U2** at the vertex at the bottom-right corner of the frame. Name this boundary condition `Roller` and the associated set `right`.
7. In the Model Tree, click mouse button 3 on the **BCs** container and select **Manager** from the menu that appears.

   Abaqus/CAE displays the **Boundary Condition Manager**. The manager indicates that the boundary conditions are `Created` (activated) in the initial step and are `Propagated from base state` (continue to be active) in the analysis step `Apply load`.

   Tip: To view the title of a column in its entirety, expand its width by dragging the dividing line between the column headings.

8. Click **Dismiss** to close the **Boundary Condition Manager**.

Applying a load to the frame

Now that you have constrained the frame, you can apply a load to the bottom of the frame.
In Abaqus the term *load* (as in the Load module in Abaqus/CAE) generally refers to anything that induces a change in the response of a structure from its initial state, including:

- concentrated forces,
- pressures,
- nonzero boundary conditions,
- body loads, and
- temperature (with thermal expansion of the material defined).

Sometimes the term load refers specifically to force-type quantities (as in the **Load Manager** of the Load module); for example, concentrated forces, pressures, and body loads but not boundary conditions or temperature. The intended meaning of the term should be clear from the context of the discussion.

In this simulation, a concentrated force of 250 kN is applied in the negative 2-direction to the midspan of the frame; the load is applied during the linear perturbation step you created earlier. In reality there is no such thing as a concentrated, or point, load; the load will always be applied over some finite area. However, if the area being loaded is small, it is an appropriate idealization to treat the load as a concentrated load.

1. In the Model Tree, click mouse button 3 on the **Loads** container and select **Manager** from the menu that appears.

   The **Load Manager** appears.

2. At the bottom of the **Load Manager**, click **Create**.

   The **Create Load** dialog box appears.

3. In the **Create Load** dialog box:
   a. Name the load `Force`.
   b. From the list of steps, select `Apply load` as the step in which the load will be applied.
   c. In the **Category** list, accept **Mechanical** as the default category selection.
   d. In the **Types for Selected Step** list, accept the default selection of **Concentrated force**.
   e. Click **Continue**.

   Abaqus/CAE displays prompts in the prompt area to guide you through the procedure. You are asked to select a region to which the load will be applied.

   As with boundary conditions, the region to which the load will be applied can be selected either directly in the viewport or from a list of existing sets. As before, you will select the region directly in the viewport.

4. In the viewport, select the vertex at the bottom center of the frame as the region where the load will be applied. Name the associated set `center`.
5. Click mouse button 2 in the viewport or click **Done** in the prompt area to indicate that you have finished selecting regions.

   The **Edit Load** dialog box appears.

6. In the dialog box:
   a. Enter a magnitude of `-250000.0` for **CF2**.
   b. Click **OK** to create the load and to close the dialog box.

      Abaqus/CAE displays a downward-pointing arrow at the vertex to indicate that the load is applied in the negative 2-direction.

7. Examine the **Load Manager** and note that the new load is `Created` (activated) in the analysis step `Apply load`.
8. Click **Dismiss** to close the **Load Manager**.

Meshing the model

You will now generate the finite element mesh. You can choose the meshing technique that Abaqus/CAE will use to create the mesh, the element shape, and the element type.
The meshing technique for one-dimensional regions (such as the ones in this example) cannot be changed, however. Abaqus/CAE uses a number of different meshing techniques. The default meshing technique assigned to the model is indicated by the color of the model that is displayed when you enter the Mesh module; if Abaqus/CAE displays the model in orange, it cannot be meshed without assistance from you.

In this section:

- *Assigning an Abaqus element type*
- *Creating the mesh*

Assigning an Abaqus element type

In this section you will assign a particular Abaqus element type to the model. Although you will assign the element type now, you could also wait until after the mesh has been created.

Two-dimensional truss elements will be used to model the frame. These elements are chosen because truss elements, which carry only tensile and compressive axial loads, are ideal for modeling pin-jointed frameworks such as this overhead hoist.

1. In the Model Tree, expand the **Frame** item underneath the **Parts** container if it is not already expanded. Then double-click **Mesh** in the list that appears.

   Abaqus/CAE switches to the Mesh module. The Mesh module functionality is available only through menu bar items or toolbox icons.

2. From the main menu bar, select **Mesh->Element Type**.
3. If you created the set named **all** while assigning section properties, click **Sets** in the right side of the prompt area and select **all** from the **Region Selection** dialog box. Otherwise, drag the mouse to create a box that selects the entire frame as the region to be assigned an element type and click **Done** in the prompt area when you are finished.

   The **Element Type** dialog box appears.

4. In the dialog box, select the following:

   - **Standard** as the **Element Library** selection (the default).
   - **Linear** as the **Geometric Order** (the default).
   - **Truss** as the **Family** of elements.

5. In the lower portion of the dialog box, examine the element shape options. A brief description of the default element selection is available at the bottom of each tabbed page.

   Since the model is a two-dimensional truss, only two-dimensional truss element types are shown on the **Line** tabbed page. A description of the element type T2D2 appears at the bottom of the dialog box. Abaqus/CAE will now associate T2D2 elements with the elements in the mesh.

6. Click **OK** to assign the element type and to close the dialog box.
7. In the prompt area, click **Done** to end the procedure.

Creating the mesh

Basic meshing is a two-stage operation: first you seed the edges of the part instance, and then you mesh the part instance. You select the number of seeds based on the desired element size or on the number of elements that you want along an edge, and Abaqus/CAE places the nodes of the mesh at the seeds whenever possible. For this problem you will create one element on each bar of the hoist.



Tip: You can display the node and element numbers within the Mesh module by selecting **View->Part Display Options** from the main menu bar. Toggle on **Show node labels** and **Show element labels** in the **Mesh** tabbed page of the **Part Display Options** dialog box that appears.

1. From the main menu bar, select **Seed->Part** to seed the part instance.

   Note:

   You can gain more control of the resulting mesh by seeding each edge of the part instance individually, but it is not necessary for this example.

   The **Global Seeds** dialog box appears. The dialog box displays the default element size that Abaqus/CAE will use to seed the part instance. This default element size is based on the size of the part instance. A relatively large seed value will be used so that only one element will be created per region.
2. In the **Global Seeds** dialog box, specify an approximate global element size of `1.0`, and click **OK** to create the seeds and to close the dialog box.
3. From the main menu bar, select **Mesh->Part** to mesh the part instance.
4. From the buttons in the prompt area, click **Yes** to confirm that you want to mesh the part instance.

Creating an analysis job

Now that you have configured your analysis, you will create a job that is associated with your model.

1. In the Model Tree, double-click the **Jobs** container to create a job.

   Abaqus/CAE switches to the Job module, and the **Create Job** dialog box appears with a list of the models in the model database. When you are finished defining your job, the **Jobs** container will display a list of your jobs.

2. Name the job `Frame`, and click **Continue**.

   The **Edit Job** dialog box appears.

3. In the **Description** field, type `Two-dimensional overhead hoist frame`.
4. Click **OK** to accept all other default job settings in the job editor and to close the dialog box.

Checking the model

Having generated the model for this simulation, you are ready to run the analysis. Unfortunately, it is possible to have errors in the model because of incorrect or missing data. You should perform a data check analysis first before running the simulation.

Run a data check analysis

1. In the Model Tree, expand the **Jobs** container. Click mouse button 3 on the job named **Frame**, and select **Data Check** from the menu that appears to submit your job for a data check analysis.
2. After you submit your job, information appears next to the job name indicating the job's status. The status of the overhead hoist problem indicates one of the following conditions:

   - **Check Submitted** while the model is being submitted for a data check.
   - **Check Running** while Abaqus performs a data check on the model.
   - **Check Completed** when the data check has completed successfully.
   - **Submitted** while the job is being submitted for a full analysis.
   - **Running** while Abaqus analyzes the model.
   - **Completed** when the full analysis is complete, and the output has been written to the output database.
   - **Aborted** if Abaqus/CAE finds a problem with the input file or the analysis ends abnormally. In addition, Abaqus/CAE reports the problem in the message area (see *Figure 1*).

Monitor the status of a job

During the analysis, Abaqus/Standard sends information to Abaqus/CAE to allow you to monitor the progress of the job. Information from the status, data, log, and message files appears in the **Job Monitor** dialog box. You can browse these files in the tabbed windows in the bottom half of the **Job Monitor** dialog box, and you can search their contents. Select the desired file tab, enter a search string in the **Text to find** field, and click **Next** or **Previous** to step through the file from one hit to the next. Toggle on **Match case** to perform a case-sensitive search.

1. In the Model Tree, click mouse button 3 on the job named **Frame** and select **Monitor** from the menu that appears to open the **Job Monitor** dialog box.

   The **Job Monitor** dialog box appears.

2. The top half of the dialog box displays the information available in the status (`.sta`) file that Abaqus creates for the analysis. This file contains a brief summary of the progress of an analysis and is described in *About Output*. The bottom half of the dialog box allows you to view information about the analysis.

   - Click the **Log** tab to display the start and end times for the analysis that appear in the log (`.log`) file.
   - Click the **Errors** and **Warnings** tabs to display the errors or the warnings that appear in the data (`.dat`) and message (`.msg`) files. If a particular region of the model is causing the error or warning, a node or element set will be created automatically that contains that region. The name of the node or element set appears with the error or warning message, and you can view the set using display groups in the Visualization module.

     It will not be possible to perform the analysis until the causes of any error messages are corrected. In addition, you should always investigate the reason for any warning messages to determine whether corrective action is needed or whether such messages can be ignored safely.

     Abaqus limits the number of error and warning messages that appear in the job monitor (by default these limits are 10 error messages and 50 warning messages). If the message limits are exceeded,

information regarding the additional errors and warnings can be obtained from the printed output files themselves. See *Job customization parameters* for details on changing the default message limits.

- Click the **Output** tab to display a record of each output data entry as it is written to the output database.
- As the analysis proceeds and Abaqus creates the data, message, and status files, Abaqus/CAE activates the **Data File**, **Message File**, and **Status File** tabs and displays the contents of each file in the corresponding tabbed page. You can click any of these tabs to browse or search the file for additional error and warning messages.



Note:

Although Abaqus/CAE updates the contents of the **Data File**, **Message File**, and **Status File** tabbed pages periodically as the analysis runs, the data might not always be synchronized with the latest data in the files.

3. Click **Dismiss** to close the **Job Monitor** dialog box.

Running the analysis

Make any necessary corrections to your model. When the data check analysis completes with no error messages, run the analysis itself. To do this, click mouse button 3 on the job named **Frame** and select **Continue** from the menu that appears.



Note:

Performing a data check analysis results in the creation of job files. When you submit a job for analysis and files of the same name already exist, Abaqus/CAE displays a dialog box asking if it is OK to overwrite the job files. Click **OK** to continue.

You should always perform a data check analysis before running a simulation to ensure that the model has been defined correctly and to check that there is enough disk space and memory available to complete the analysis. However, it is possible to combine the data check and analysis phases of the simulation by clicking mouse button 3 on the job name in the **Jobs** container and selecting **Submit** from the menu that appears.

If a simulation is expected to take a substantial amount of time, it may be convenient to run it in a batch queue by selecting **Queue** as the **Run Mode** in the **Edit Job** dialog box. (The availability of such a queue depends on the queue definition settings in your Abaqus environment file. If you have any questions, refer to *Defining analysis batch queues* or ask your system administrator about defining analysis batch queues on your system.)

Postprocessing

Graphical postprocessing is important because of the great volume of data created during a simulation. The Visualization module of Abaqus/CAE (also licensed separately as Abaqus/Viewer) allows you to view the results graphically using a variety of methods, including deformed shape plots, contour plots, vector plots, animations, and *X–Y* plots. In addition, it allows you to create tabular reports of the output data. All of these methods are discussed in this guide. For more information on any of the postprocessing features discussed in this guide, consult *Viewing results*.

In this section:

- *Postprocessing: overhead hoist*
- *The Results Tree*
- *Customizing an undeformed shape plot*
- *Displaying and customizing a deformed shape plot*
- *Checking the model with Abaqus/CAE*
- *Generating tabular data reports*
- *Reviewing tabular data reports*

Postprocessing: overhead hoist

For this example you will use the Visualization module to do some basic model checks and to display the deformed shape of the frame.

When the job completes successfully, you are ready to view the results of the analysis with the Visualization module. In the Model Tree, click mouse button 3 on the job named **Frame** and select **Results** from the menu that appears to enter the Visualization module. Abaqus/CAE opens the output database created by the job and displays the undeformed model shape, as shown in *Figure 1*.

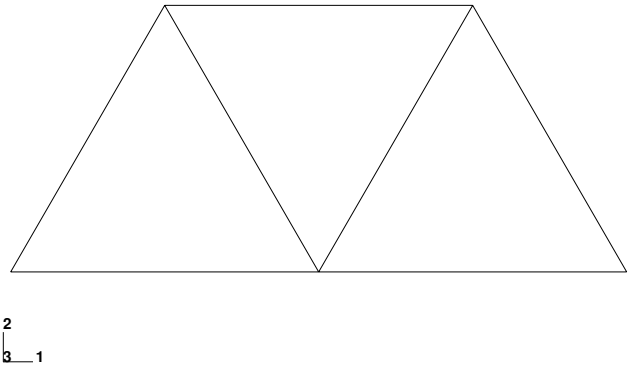


Figure 1: Undeformed model shape.

You can also enter the Visualization module by selecting **Visualization** from the **Module** list located in the context bar. Select **File->Open**, select `Frame.odb` from the list of available output database files, and click **OK**.

You can choose to display the title block and state block at the bottom of the viewport; these blocks are not shown in *Figure 1*. The title block at the bottom of the viewport indicates the following:

- The description of the model (from the job description).
- The name of the output database (from the name of the analysis job).
- The product name (Abaqus/Standard or Abaqus/Explicit) and release used to generate the output database.
- The date the output database was last modified.

The state block at the bottom of the viewport indicates the following:

- Which step is being displayed.
- The increment within the step.
- The step time.

The view orientation triad indicates the orientation of the model in the global coordinate system. The 3D compass located in the upper-right corner of the viewport allows you to manipulate the view directly.

You can suppress the display of and customize the title block, state block, view orientation triad, and 3D compass by selecting **Viewport->Viewport Annotation Options** from the main menu bar (for example, many of the figures in this guide do not include the title block or the compass).

The Results Tree

You will use the Results Tree to query the components of the model. The Results Tree allows easy access to the history output contained in an output database file for the purpose of creating *X–Y* plots and also to groups of elements, nodes, and surfaces based on set names, material and section assignment, etc. for the purposes of verifying the model and also controlling the viewport display.

The Results Tree will be used more extensively in later examples to illustrate the *X–Y* plotting capability and manipulating the display using display groups.

1. In the left side of the main window, click the **Results** tab to switch to the Results Tree if it is not already visible.
2. All output database files that are open in a given postprocessing session are listed underneath the **Output Databases** container. Expand this container and then expand the container for the output database named `Frame.odb`.
3. Expand the **Materials** container, and click the material named **STEEL**.

   All elements are highlighted in the viewport because only one material assignment was used in this analysis.

Customizing an undeformed shape plot

You will now use the plot options to enable the display of node and element numbering.
Plot options that are common to all plot types (undeformed, deformed, contour, symbol, and material orientation) are set in a single dialog box. The contour, symbol, and material orientation plot types have additional options, each specific to the given plot type.

Display node numbers

1. From the main menu bar, select **Options->Common**; or use the  tool in the toolbox.
   The **Common Plot Options** dialog box appears.
2. Click the **Labels** tab.
3. Toggle on **Show node labels**.
4. Click **Apply**.
   Abaqus/CAE applies the change and keeps the dialog box open.
   The customized undeformed plot is shown in *Figure 1* (your node numbers may be different).

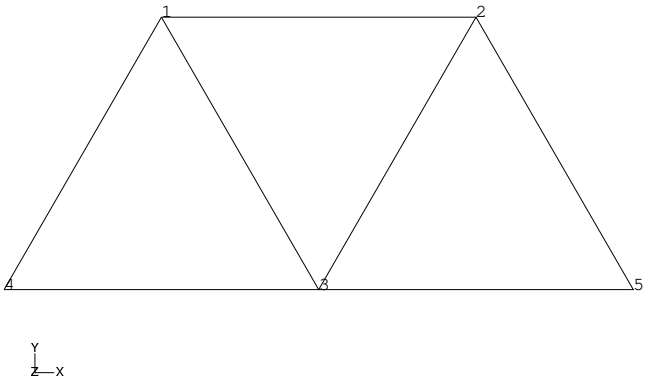


Figure 1: Node number plot.

Display element numbers

1. In the **Labels** tabbed page of the **Common Plot Options** dialog box, toggle on **Show element labels**.
2. Click **OK**.
   Abaqus/CAE applies the change and closes the dialog box.
   The resulting plot is shown in *Figure 2* (your element numbers may be different).

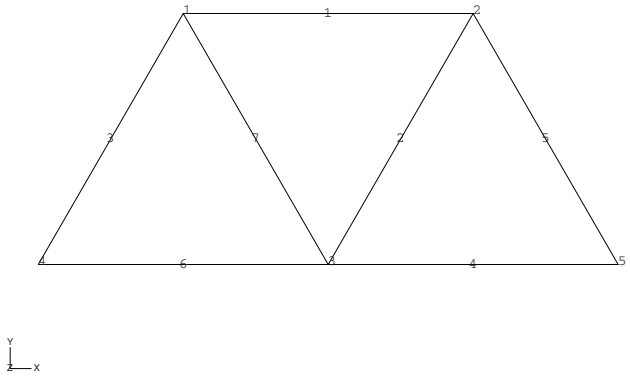


Figure 2: Node and element number plot.

Remove the node and element labels before proceeding. To disable the display of node and element numbers, repeat the above procedure and, under **Labels**, toggle off **Show node labels** and **Show element labels**.

Displaying and customizing a deformed shape plot

You will now display the deformed model shape and use the plot options to change the deformation scale factor. You will also superimpose the undeformed model shape on the deformed model shape.

From the main menu bar, select **Plot->Deformed Shape**; or use the  tool in the toolbox. Abaqus/CAE displays the deformed model shape, as shown in *Figure 1*.

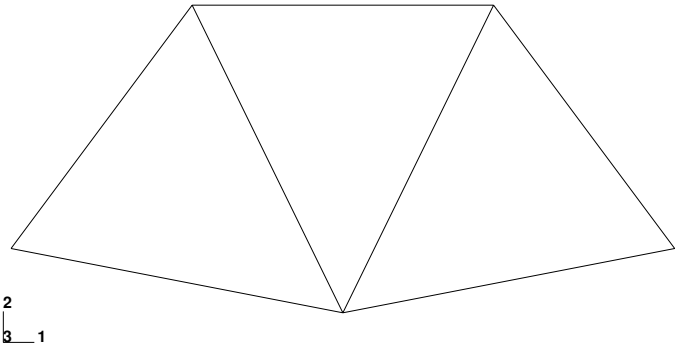


Figure 1: Deformed model shape.

For small-displacement analyses (the default formulation in Abaqus/Standard) the displacements are scaled automatically to ensure that they are clearly visible. The scale factor is displayed in the state block. In this case the displacements have been scaled by a factor of 42.83.

Change the deformation scale factor

1. From the main menu bar, select **Options->Common**; or use the  tool in the toolbox.
2. From the **Common Plot Options** dialog box, click the **Basic** tab if it is not already selected.
3. From the **Deformation Scale Factor** area, toggle on **Uniform** and enter `10.0` in the **Value** field.
4. Click **Apply** to redisplay the deformed shape.

   The state block displays the new scale factor.
5. To return to automatic scaling of the displacements, repeat the above procedure and, in the **Deformation Scale Factor** field, toggle on **Auto-compute**.
6. Click **OK** to close the **Common Plot Options** dialog box.

Superimpose the undeformed model shape on the deformed model shape

1. Click the **Allow Multiple Plot States**  tool in the toolbox to allow multiple plot states in the viewport; then click the  tool or select **Plot->Undeformed Shape** to add the undeformed shape plot to the existing deformed plot in the viewport.

   By default, Abaqus/CAE plots the deformed model shape in green and the (superimposed) undeformed model shape in a translucent white.
2. The plot options for the superimposed image are controlled separately from those of the primary image.

   From the main menu bar, select **Options->Superimpose**; or use the  tool in the toolbox to change the edge style of the superimposed (i.e., undeformed) image.
3. From the **Superimpose Plot Options** dialog box, click the **Color & Style** tab.

4. In the **Color & Style** tabbed page, select the dashed edge style.
5. Click **OK** to close the **Superimpose Plot Options** dialog box and to apply the change.

   The plot is shown in *Figure 2*. The undeformed model shape appears with a dashed edge style.

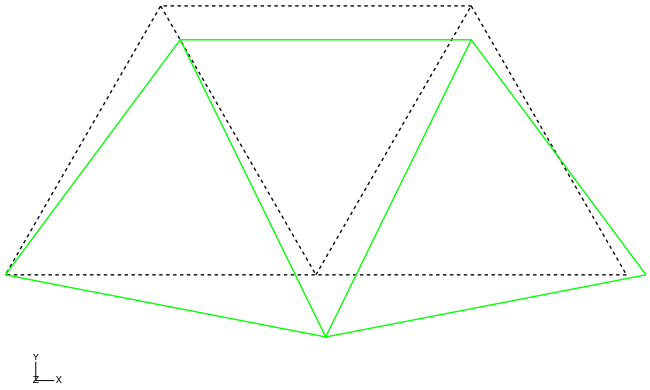


Figure 2: Undeformed and deformed model shapes.

Checking the model with Abaqus/CAE

You can use Abaqus/CAE to check that the model is correct before running the simulation. You have already learned how to draw plots of the model and to display the node and element numbers. These are useful tools for checking that Abaqus is using the correct mesh.

The boundary conditions applied to the overhead hoist model can also be displayed and checked in the Visualization module.

1. Click the  tool in the toolbox to disable multiple plot states in the viewport.
2. Display the undeformed model shape, if it is not displayed already.
3. From the main menu bar, select **View->ODB Display Options**.
4. In the **ODB Display Options** dialog box, click the **Entity Display** tab.
5. Toggle on **Show boundary conditions**.
6. Click **OK**.

   Abaqus/CAE displays symbols to indicate the applied boundary conditions, as shown in *Figure 1*.

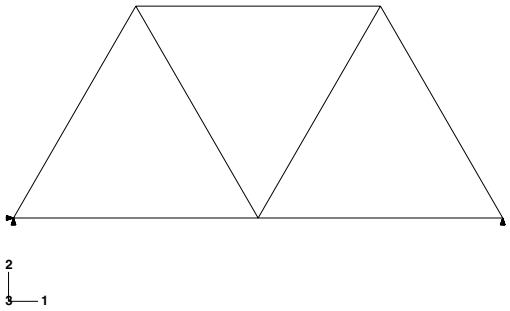


Figure 1: Applied boundary conditions on the overhead hoist.

Generating tabular data reports

In addition to the graphical capabilities described above, Abaqus/CAE allows you to write data to a text file in a tabular format. This feature is a convenient alternative to writing tabular output to the data (`.dat`) file. Output generated this way has many uses; for example, it can be used in written reports. In this problem you will generate a report containing the element stresses, nodal displacements, and reaction forces.

1. From the main menu bar, select **Report->Field Output**.
2. In the **Variable** tabbed page of the **Report Field Output** dialog box, accept the default position labeled **Integration Point**. Click the triangle next to **S: Stress components** to expand the list of available variables. From this list, toggle on **S11**.
3. In the **Setup** tabbed page, name the report `Frame.rpt`. In the **Data** region at the bottom of the page, toggle off **Column totals**.
4. Click **Apply**.

   The element stresses are written to the report file.
5. In the **Variable** tabbed page of the **Report Field Output** dialog box, change the position to **Unique Nodal**. Toggle off **S: Stress components**, and select **U1** and **U2** from the list of available **U: Spatial displacement** variables.
6. Click **Apply**.

   The nodal displacements are appended to the report file.
7. In the **Variable** tabbed page of the **Report Field Output** dialog box, toggle off **U: Spatial displacement**, and select **RF1** and **RF2** from the list of available **RF: Reaction force** variables.
8. In the **Data** region at the bottom of the **Setup** tabbed page, toggle on **Column totals**.
9. Click **OK**.

   The reaction forces are appended to the report file, and the **Report Field Output** dialog box closes.

Reviewing tabular data reports

Open the file `Frame.rpt` in a text editor. The contents of this file are shown below. Your node and element numbering may be different. Very small values may also be calculated differently, depending on your system.

Stress output:

```
Field Output Report

Source 1
---------

   ODB: Frame.odb
   Step: Apply load
   Frame: Increment      1: Step Time =   2.2200E-16

Loc 1 : Integration point values from source 1

Output sorted by column "Element Label".

Field Output reported at integration points for part: FRAME-1

         Element             Int           S.S11
           Label              Pt          @Loc 1
-------------------------------------------------
               1               1    -294.116E+06
               2               1     294.116E+06
               3               1    -294.116E+06
               4               1     147.058E+06
               5               1    -294.116E+06
               6               1     147.058E+06
               7               1     294.116E+06


  Minimum                            -294.116E+06
      At Element                                5

          Int Pt                                1
  Maximum                             294.116E+06
      At Element                                7

          Int Pt                                1
```

Displacement output:

```
Field Output Report

Source 1
---------

   ODB: Frame.odb
   Step: Apply load
   Frame: Increment      1: Step Time =   2.2200E-16

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".

Field Output reported at nodes for part: FRAME-1
```

```
            Node            U.U1            U.U2
           Label         @Loc 1          @Loc 1
-------------------------------------------------
               1     1.47058E-03    -2.54712E-03
               2    -46.3233E-12    -2.54712E-03
               3     735.291E-06    -4.66972E-03
               4             -0.         -5.E-33
               5     1.47058E-03         -5.E-33


  Minimum          -46.3233E-12    -4.66972E-03

         At Node               2               3
  Maximum            1.47058E-03         -5.E-33

         At Node               5               5
```

Reaction force output:

```
Field Output Report

Source 1
---------

   ODB: Frame.odb
   Step: Apply load
   Frame: Increment      1: Step Time =   2.2200E-16

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".

Field Output reported at nodes for part: FRAME-1

            Node          RF.RF1          RF.RF2
           Label         @Loc 1          @Loc 1
-------------------------------------------------
               1              0.              0.
               2              0.              0.
               3              0.              0.
               4     454.747E-15          5.E+03
               5              0.          5.E+03


  Minimum                     0.              0.
         At Node               5               3

  Maximum            454.747E-15          5.E+03
         At Node               4               5

           Total     454.747E-15         10.E+03
```

Are the nodal displacements and peak stresses in the individual members reasonable for this hoist and these applied loads?

It is always a good idea to check that the results of the simulation satisfy basic physical principles. In this case check that the external forces applied to the hoist and the reaction forces sum to zero in both the vertical and horizontal directions.

What nodes have vertical forces applied to them? What nodes have horizontal forces? Do the results from your simulation match those shown here?

Rerunning the analysis using Abaqus/Explicit

We will rerun the same analysis in Abaqus/Explicit for comparison.

This time we are interested in the dynamic response of the hoist to the same load applied suddenly at the midspan. Before continuing, click the **Model** tab in the left side of the main window to switch to the Model Tree. Click mouse button 3 on **Model-1** in the Model Tree and select **Copy Model** from the menu that appears to copy the existing model to a new model named `Explicit`. Make all subsequent changes to the `Explicit` model (you may want to collapse the original model to avoid confusion). You will need to replace the static step with an explicit dynamic step, modify the output requests and the material definition, and change the element library before you can resubmit the job.

In this section:

- *Replacing the analysis step*
- *Modifying the output requests*
- *Modifying the material definition*
- *Changing the element library and submitting the job for analysis*

Replacing the analysis step

The step definition must change to reflect a dynamic, explicit analysis.

1. In the Model Tree, expand the **Steps** container. Click mouse button 3 on the step named **Apply load**, and select **Replace** from the menu that appears.
2. In the **Replace Step** dialog box, select **Dynamic, Explicit** from the list of available **General** procedures. Click **Continue**.

   Model attributes such as boundary conditions, loads, and contact interactions are retained when replacing a step. Model attributes that cannot be converted will be deleted. In this simulation all necessary model attributes will be retained.
3. In the **Basic** tabbed page of the **Edit Step** dialog box, enter the step description `10 kN central load, suddenly applied` and set the time period of the step to `0.01` s.

Modifying the output requests

Because this is a dynamic analysis in which the transient response of the frame is of interest, it is helpful to have the displacements of the center point written as history output. Displacement history output can be requested only for a preselected set.
You will use the set named **center**, which was created earlier while assigning the load.

Create a set (if necessary)

1. In the Model Tree, expand the **Assembly** container and double-click the **Sets** item.

   The **Create Set** dialog box appears.

2. Name the set `center`, accept the default selection of **Geometry**, and click **Continue**.

   Note:

   For meshed parts you can define sets that are based on either the geometry or the mesh. If you modify the mesh, you must redefine a mesh-based set. However, a geometry-based set will update automatically. For unmeshed parts only geometry-based sets are available.

3. In the viewport, select the point at the center of the bottom edge of the truss. In the prompt area, click **Done** when you are finished.

Add displacements to the history output request

1. In the Model Tree, click mouse button 3 on the **History Output Requests** container and select **Manager** from the menu that appears.
2. In the **History Output Requests Manager** dialog box that appears, click **Edit**.

   The history output editor appears.

3. Under the **Domain** field, select **Set**. Abaqus automatically provides a list of all the sets created for a given model. Choose the set named **center**.
4. Under the **Frequency** field, select **Every n time increments** and set the value of **n** to `1` to write the output at every increment.
5. In the **Output Variables** region, toggle off the **Energy** output and click the arrow to the left of the **Displacement/Velocity/Acceleration** category to reveal history output options for translations and rotations.
6. Toggle on **UT, Translations** to have the displacements for the selected set be written as history output to the output database file.
7. Click **OK** to save your changes and to close the dialog box. Dismiss the **History Output Requests Manager**.

Modifying the material definition

Since Abaqus/Explicit performs a dynamic analysis, a complete material definition requires that you specify the material density. For this problem assume the density is equal to 7800 kg/m^3.

1. In the Model Tree, expand the **Materials** container and double-click **Steel**.
2. In the material editor, select **General->Density** and type a value of `7800` for the density.
3. Click **OK** to close the **Edit Material** dialog box.

Changing the element library and submitting the job for analysis

The elements available in Abaqus/Explicit are a subset of those available in Abaqus/Standard, so you must change the element library from which the elements are selected to the explicit element library.
After changing the element library, you will create and run a new job for the Abaqus/Explicit analysis.

To ensure that you are using a valid element type in your analysis, Abaqus/CAE automatically filters the available element types according to the selected element library.

• *Finite Elements and Rigid Bodies*

Change the element library

1. In the Model Tree, expand the **Frame** item underneath the **Parts** container. Then double-click **Mesh** in the list that appears.

   Abaqus/CAE switches to the Mesh module.
2. From the main menu bar, select **Mesh->Element Type**, select the set **all** (or pick the entire frame in the viewport), and change the **Element Library** selection to **Explicit**.
3. Click **OK** to accept the new element type.

Create and run the new job

1. In the Model Tree, double-click the **Jobs** container.
2. Name the new job `expFrame`, and select **Explicit** as the source model.
3. Submit the job.

Postprocessing the dynamic analysis results

For the static linear perturbation analysis done in Abaqus/Standard you examined the deformed shape as well as stress, displacement, and reaction force output. For the Abaqus/Explicit analysis you can similarly examine the deformed shape and generate field data reports.
Because this is a dynamic analysis, you should also examine the transient response resulting from the loading. You will do this by animating the time history of the deformed model shape and plotting the displacement history of the bottom center node in the truss.

Plot the deformed shape of the model. For large-displacement analyses (the default formulation in Abaqus/Explicit) the displaced shape scale factor has a default value of 1. Change the **Deformation Scale Factor** to 20 so that you can more easily see the deformation of the truss.

Create a time-history animation of the deformed model shape

1. From the main menu bar, select **Animate->Time History**; or use the  tool in the toolbox.

   The time history animation begins in a continuous loop at its fastest speed. Abaqus/CAE displays the movie player controls in the right side of the context bar (immediately above the viewport).

2. From the main menu bar, select **Options->Animation**; or use the animation options  tool in the toolbox (located directly underneath the  tool).

   The **Animation Options** dialog box appears.

3. Change the **Mode** to **Play Once**, and slow the animation down by moving the **Frame Rate** slider.
4. You can use the animation controls to start, pause, and step through the animation. From left to right of *Figure 1*, these controls perform the following functions: **play/pause**, **first**, **previous**, **next**, and **last**.

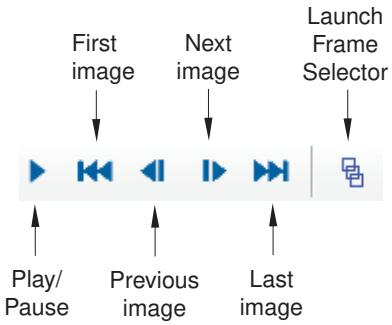


Figure 1: Postprocessing animation controls.

Create an X–Y plot of the vertical displacement for a node

The truss responds dynamically to the load. You can confirm this by plotting the vertical displacement history of the node set `Center`.

You can create X–Y curves from either history or field data stored in the output database (`.odb`) file. X–Y curves can also be read from an external file or they can be typed into the Visualization module interactively. Once curves have been created, their data can be further manipulated and plotted to the screen in graphical form. In this example you will create and plot the curve using history data.

1. In the Results Tree, expand the **History Output** container underneath the output database named `expFrame.odb`.
2. From the list of available history output, double-click `Spatial displacement: U2 at Node x in NSET CENTER`.

   Abaqus/CAE plots the vertical displacement at the center node along the bottom of the truss, as shown in *Figure 2*.

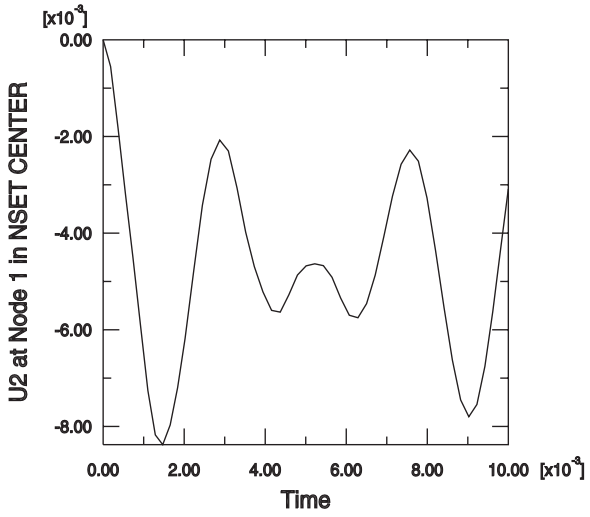


Figure 2: Vertical displacement at the midspan of the truss.

Note: The chart legend has been suppressed and the axis labels modified in this figure. Many *X–Y* plot options are directly accessible by double-clicking the appropriate regions of the viewport.

To enable direct object actions, however, you must first click in the prompt area to cancel the current procedure (if necessary). To suppress the legend, double-click it in the viewport to open the **Chart Legend Options** dialog box. In the **Contents** tabbed page of this dialog box, toggle off **Show legend**. To modify the axis labels, double-click either axis to open the **Axis Options** dialog box, and edit the axis titles as indicated in *Figure 2*.

Exiting Abaqus/CAE

Save your model database file; then select **File->Exit** from the main menu bar to exit Abaqus/CAE.

Comparison of implicit and explicit procedures

Abaqus/Standard and Abaqus/Explicit are capable of solving a wide variety of problems. The characteristics of implicit and explicit procedures determine which method is appropriate for a given problem.
For those problems that can be solved with either method, the efficiency with which the problem can be solved can determine which product to use. Understanding the characteristics of implicit and explicit procedures will help you answer this question.

In this section:

- *Choosing between implicit and explicit analysis*
- *Cost of mesh refinement in implicit and explicit analyses*

Choosing between implicit and explicit analysis

For many analyses it is clear whether Abaqus/Standard or Abaqus/Explicit should be used.
For example, as demonstrated in *Nonlinearity*, Abaqus/Standard is more efficient for solving smooth nonlinear problems; on the other hand, Abaqus/Explicit is the clear choice for a wave propagation analysis. There are, however, certain static or quasi-static problems that can be simulated well with either program. Typically, these are problems that usually would be solved with Abaqus/Standard but may have difficulty converging because of contact or material complexities, resulting in a large number of iterations. Such analyses are expensive in Abaqus/Standard because each iteration requires a large set of linear equations to be solved.

Whereas Abaqus/Standard must iterate to determine the solution to a nonlinear problem, Abaqus/Explicit determines the solution without iterating by *explicitly* advancing the kinematic state from the end of the previous increment. Even though a given analysis may require a large number of time increments using the explicit method, the analysis can be more efficient in Abaqus/Explicit if the same analysis in Abaqus/Standard requires many iterations.

Another advantage of Abaqus/Explicit is that it requires much less disk space and memory than Abaqus/Standard for the same simulation. For problems in which the computational cost of the two programs may be comparable, the substantial disk space and memory savings of Abaqus/Explicit make it attractive.

Table 1 lists the key differences between the analysis products, which are discussed in detail in the relevant chapters in this guide.

Table 1: Key differences between Abaqus/Standard and Abaqus/Explicit.

Quantity	Abaqus/Standard	Abaqus/Explicit
Element library	Offers an extensive element library.	Offers an extensive library of elements well suited for explicit analyses. The elements available are a subset of those available in Abaqus/Standard.
Analysis procedures	General and linear perturbation procedures are available.	General procedures are available.
Material models	Offers a wide range of material models.	Similar to those available in Abaqus/Standard; a notable difference is that failure material models are allowed.
Contact formulation	Has a robust capability for solving contact problems.	Has a robust contact functionality that readily solves even the most complex contact simulations.
Solution technique	Uses a stiffness-based solution technique that is unconditionally stable.	Uses an explicit integration solution technique that is conditionally stable.
Disk space and memory	Due to the large numbers of iterations possible in an increment, disk space and memory usage can be large.	Disk space and memory usage is typically much smaller than that for Abaqus/Standard.

Cost of mesh refinement in implicit and explicit analyses

Using the explicit method, the computational cost is proportional to the number of elements and roughly inversely proportional to the smallest element dimension. Mesh refinement, therefore, increases the computational cost by increasing the number of elements and reducing the smallest element dimension. As an example, consider a three-dimensional model with uniform, square elements. If the mesh is refined by a factor of two in all three directions, the computational cost increases by a factor of $2 \times 2 \times 2$ as a result of the increase in the number of elements and by a factor of 2 as a result of the decrease in the smallest element dimension. The total computational cost of the analysis increases by a factor of 2^4, or 16, by refining the mesh. Disk space and memory requirements are proportional to the number of elements with no dependence on element dimensions; thus, these requirements increase by a factor of 8.

Whereas predicting the cost increase with mesh refinement for the explicit method is rather straightforward, cost is more difficult to predict when using the implicit method. The difficulty arises from the problem-dependent relationship between element connectivity and solution cost, a relationship that does not exist in the explicit method. Using the implicit method, experience shows that for many problems the computational cost is roughly proportional to the square of the number of degrees of freedom. Consider the same example of a three-dimensional model with uniform, square elements. Refining the mesh by a factor of two in all three directions increases the number of degrees of freedom by approximately 2^3, causing the computational cost to increase by a factor of roughly $(2^3)^2$, or 64. The disk space and memory requirements increase in the same manner, although the actual increase is difficult to predict.

The explicit method shows great cost savings over the implicit method as the model size increases, as long as the mesh is relatively uniform. *Figure 1* illustrates the comparison of cost versus model size using the explicit and implicit methods. For this problem the number of degrees of freedom scales with the number of elements.

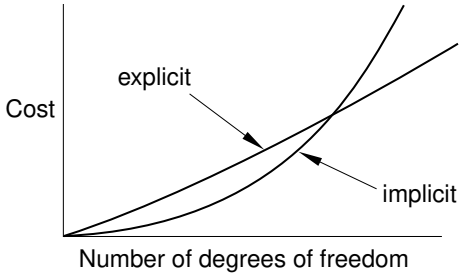


Figure 1: Cost versus model size in using the explicit and implicit methods.

Summary

You can use Abaqus for preprocessing (Abaqus/CAE), simulation (Abaqus/Standard and Abaqus/Explicit), and postprocessing (Abaqus/CAE).

- Abaqus/CAE can be used to create a complete Abaqus analysis model. The analysis product (Abaqus/Standard or Abaqus/Explicit) reads the input file generated by Abaqus/CAE, performs the analysis, sends information to Abaqus/CAE to allow you to monitor the progress of the job, and generates an output database. You use the Visualization module to read the output database and view the results of your analysis.
- You can perform a data check analysis once you have created a model. Error and warning messages are printed to the **Job Monitor** dialog box.
- Use the Visualization module in Abaqus/CAE to verify the model geometry and boundary conditions graphically, using the output database file generated during the data check phase.
- Always check that the results satisfy basic engineering principles, such as equilibrium.
- The Visualization module of Abaqus/CAE allows you to visualize analysis results graphically in a variety of ways and also allows you to write tabular data reports.
- The choice between using implicit or explicit methods depends largely on the nature of the problem.

Finite Elements and Rigid Bodies

Finite elements and rigid bodies are the fundamental components of an Abaqus model.
Finite elements are deformable, whereas rigid bodies move through space without changing shape. While users of finite element analysis programs tend to have some understanding of what finite elements are, the general concept of rigid bodies within a finite element program may be somewhat new.

For computational efficiency Abaqus has a general rigid body capability. Any body or part of a body can be defined as a rigid body; most element types can be used in a rigid body definition (the exceptions are listed in *Rigid Body Definition*). The advantage of rigid bodies over deformable bodies is that the motion of a rigid body is described completely by no more than six degrees of freedom at a reference node. In contrast, deformable elements have many degrees of freedom and require expensive element calculations to determine the deformations. When such deformations are negligible or not of interest, modeling a component as a rigid body produces significant computational savings without affecting the overall results.

In this section:

- *Finite elements*
- *Rigid bodies*
- *Summary*

Finite elements

A wide range of elements is available in Abaqus.
This extensive element library provides you with a powerful set of tools for solving many different problems. The elements available in Abaqus/Explicit are (with a few exceptions) a subset of those available in Abaqus/Standard. This section introduces you to the five aspects of an element that influence how it behaves.

In this section:

- *Characterizing elements*
- *Continuum elements*
- *Shell elements*
- *Beam elements*
- *Truss elements*

Characterizing elements

Each element is characterized by the following:

- Family
- Degrees of freedom (directly related to the element family)
- Number of nodes
- Formulation
- Integration

Each element in Abaqus has a unique name, such as T2D2, S4R, or C3D8I. The element name identifies each of the five aspects of an element. The naming convention is explained in this chapter.

Family

Figure 1 shows the element families most commonly used in a stress analysis. One of the major distinctions between different element families is the geometry type that each family assumes.

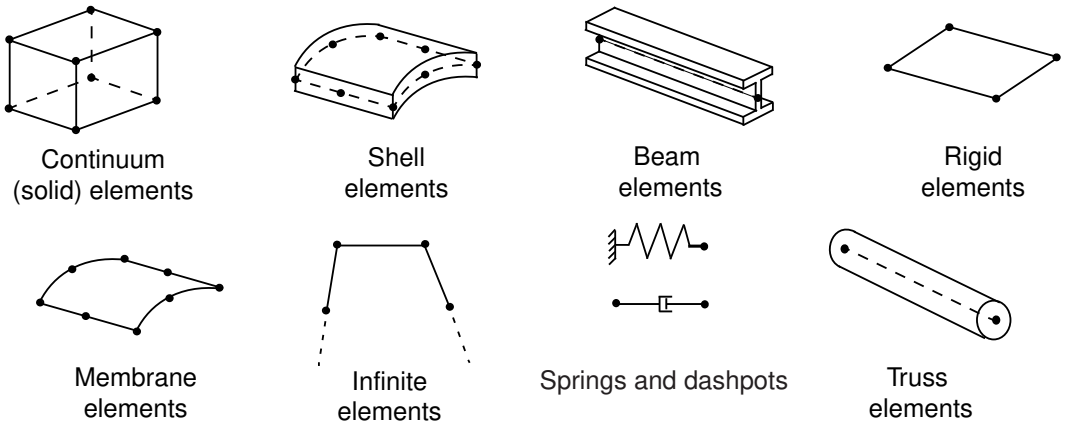


Figure 1: Commonly used element families.

The element families that you will use in this guide—continuum, shell, beam, truss, and rigid elements—are discussed in detail in other chapters. The other element families are not covered in this guide; if you are interested in using them in your models, read about them in *Abaqus Elements Guide*.

The first letter or letters of an element's name indicate to which family the element belongs. For example, the S in S4R indicates this is a shell element, while the C in C3D8I indicates this is a continuum element.

Degrees of freedom

The degrees of freedom (dof) are the fundamental variables calculated during the analysis. For a stress/displacement simulation the degrees of freedom are the translations at each node. Some element families, such as the beam and shell families, have rotational degrees of freedom as well. For a heat transfer simulation the degrees of freedom are the temperatures at each node; a heat transfer analysis, therefore, requires the use of different elements than a stress analysis, since the degrees of freedom are not the same.

The following numbering convention is used for the degrees of freedom in Abaqus:

1	Translation in direction 1
2	Translation in direction 2
3	Translation in direction 3

4	Rotation about the 1-axis
5	Rotation about the 2-axis
6	Rotation about the 3-axis
7	Warping in open-section beam elements
8	Acoustic pressure, pore pressure, or hydrostatic fluid pressure
9	Electric potential
10	Connector material flow (units of length)
11	Temperature (or normalized concentration in mass diffusion analysis) for continuum elements or temperature at the first point through the thickness of beams and shells
12+	Temperature at other points through the thickness of beams and shells

Directions 1, 2, and 3 correspond to the global 1-, 2-, and 3-directions, respectively, unless a local coordinate system has been defined at the nodes.

Axisymmetric elements are the exception, with the displacement and rotation degrees of freedom referred to as follows:

1	Translation in the *r*-direction
2	Translation in the *z*-direction
6	Rotation in the *r*–*z* plane

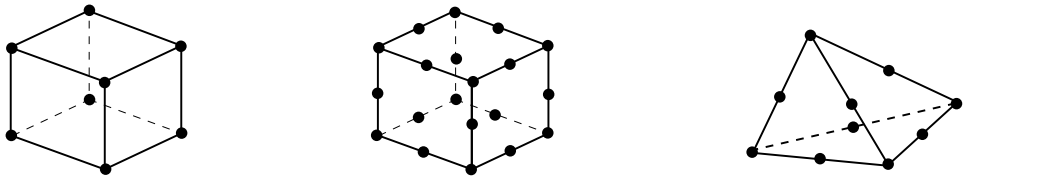
Directions *r* (radial) and *z* (axial) correspond to the global 1- and 2-directions, respectively, unless a local coordinate system has been defined at the nodes. See *Using Shell Elements* for a discussion of defining a local coordinate system at the nodes.

In this guide our attention is restricted to structural applications. Therefore, only elements with translational and rotational degrees of freedom are discussed. For information on other types of elements (for example, heat transfer elements), consult the *Abaqus Elements Guide*.

By default, Abaqus/CAE uses the alphabetical option, *x*-*y*-*z*, for labeling the view orientation triad. In general, this guide adopts the numerical option, 1-2-3, to permit direct correspondence with degree of freedom and output labeling. For more information on labeling of axes, see *Customizing the view triad*.

Number of nodes—order of interpolation

Displacements, rotations, temperatures, and the other degrees of freedom mentioned in the previous section are calculated only at the nodes of the element. At any other point in the element, the displacements are obtained by interpolating from the nodal displacements. Usually the interpolation order is determined by the number of nodes used in the element, as illustrated in the examples in *Figure 2*.



(a) Linear element (8-node brick, C3D8)

(b) Quadratic element (20-node brick, C3D20)

(c) Modified second-order element (10-node tetrahedron, C3D10M)

Figure 2: Linear brick, quadratic brick, and modified tetrahedral elements.

- Elements that have nodes only at their corners, such as the 8-node brick shown in *Figure 2*(a), use linear interpolation in each direction and are often called linear elements or first-order elements.

- Elements with midside nodes, such as the 20-node brick shown in *Figure 2*(b), use quadratic interpolation and are often called quadratic elements or second-order elements.
- Modified triangular or tetrahedral elements with midside nodes, such as the 10-node tetrahedron shown in *Figure 2*(c), use a modified second-order interpolation and are often called modified elements or modified second-order elements.

Abaqus/Standard offers a wide selection of both linear and quadratic elements. Abaqus/Explicit offers only linear elements, with the exception of the quadratic beam, quadratic tetrahedral, and modified tetrahedron and triangle elements.

Typically, the number of nodes in an element is clearly identified in its name. The 8-node brick element, as you have seen, is called C3D8; and the 8-node general shell element is called S8R. The beam element family uses a slightly different convention: the order of interpolation is identified in the name. Thus, a first-order, three-dimensional beam element is called B31, whereas a second-order, three-dimensional beam element is called B32. A similar convention is used for axisymmetric shell and membrane elements.

Formulation

An element's formulation refers to the mathematical theory used to define the element's behavior. In the absence of adaptive meshing all of the stress/displacement elements in Abaqus are based on the *Lagrangian* or *material* description of behavior: the material associated with an element remains associated with the element throughout the analysis, and material cannot flow across element boundaries. In the alternative *Eulerian* or *spatial* description, elements are fixed in space as the material flows through them. Eulerian methods are used commonly in fluid mechanics simulations. Abaqus/Standard uses Eulerian elements to model convective heat transfer. Adaptive meshing combines the features of pure Lagrangian and Eulerian analyses and allows the motion of the element to be independent of the material. Eulerian elements and adaptive meshing are not discussed in this guide.

To accommodate different types of behavior, some element families in Abaqus include elements with several different formulations. For example, the shell element family has three classes: one suitable for general-purpose shell analysis, another for thin shells, and yet another for thick shells. (These shell element formulations are explained in *Using Shell Elements*.)

Some Abaqus/Standard element families have a standard formulation as well as some alternative formulations. Elements with alternative formulations are identified by an additional character at the end of the element name. For example, the continuum, beam, and truss element families include members with a hybrid formulation in which the pressure (continuum elements) or axial force (beam and truss elements) is treated as an additional unknown; these elements are identified by the letter “H” at the end of the name (C3D8H or B31H).

Some element formulations allow coupled field problems to be solved. For example, elements whose names begin with the letter C and end with the letter T (such as C3D8T) possess both mechanical and thermal degrees of freedom and are intended for coupled thermomechanical simulations.

Several of the most commonly used element formulations are discussed later in this guide.

Integration

Abaqus uses numerical techniques to integrate various quantities over the volume of each element. Using Gaussian quadrature for most elements, Abaqus evaluates the material response at each integration point in each element. Some elements in Abaqus can use full or reduced integration, a choice that can have a significant effect on the accuracy of the element for a given problem, as discussed in detail in *Element formulation and integration*.

Abaqus uses the letter “R” at the end of the element name to distinguish reduced-integration elements (unless they are also hybrid elements, in which case the element name ends with the letters “RH”). For example, CAX4 is the 4-node, fully integrated, linear, axisymmetric solid element; and CAX4R is the reduced-integration version of the same element.

Abaqus/Standard offers both full and reduced-integration elements; Abaqus/Explicit offers only reduced-integration elements with the exception of the modified tetrahedron and triangle elements and the fully integrated first-order shell, membrane, and brick elements.

Continuum elements

Among the different element families, continuum or solid elements can be used to model the widest variety of components. Conceptually, continuum elements simply model small blocks of material in a component. Since they may be connected to other elements on any of their faces, continuum elements, like bricks in a building or tiles in a mosaic, can be used to build models of nearly any shape, subjected to nearly any loading. Abaqus has stress/displacement, nonstructural, and coupled field continuum elements; this guide will discuss only stress/displacement elements.

Continuum stress/displacement elements in Abaqus have names that begin with the letter "C." The next two letters indicate the dimensionality and usually, but not always, the active degrees of freedom in the element. The letters "3D" indicate a three-dimensional element; "AX," an axisymmetric element; "PE," a plane strain element; and "PS," a plane stress element.

The use of continuum elements is discussed further in *Using Continuum Elements*.

Three-dimensional continuum element library

Three-dimensional continuum elements can be hexahedra (bricks), wedges, pyramids, or tetrahedra. The full inventory of three-dimensional continuum elements and the nodal connectivity for each type can be found in *Three-Dimensional Solid Element Library*.

Whenever possible, hexahedral elements or second-order tetrahedral elements should be used in Abaqus. First-order tetrahedra (C3D4) have a simple, constant-strain formulation, and very fine meshes are required for an accurate solution.

Two-dimensional continuum element library

Abaqus has several classes of two-dimensional continuum elements that differ from each other in their out-of-plane behavior. Two-dimensional elements can be quadrilateral or triangular. *Figure 1* shows the three classes that are used most commonly.

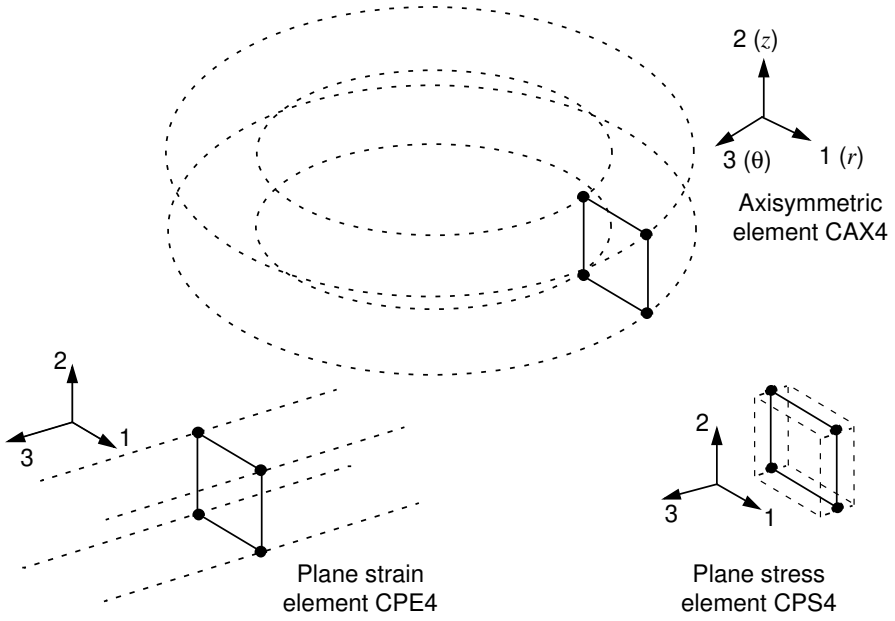


Figure 1: Plane strain, plane stress, and axisymmetric elements without twist.

Plane strain elements assume that the out-of-plane strain, ε_{33}, is zero; they can be used to model thick structures.

Plane stress elements assume that the out-of-plane stress, σ_{33}, is zero; they are suitable for modeling thin structures.

Axisymmetric elements without twist, the "CAX" class of elements, model a 360° ring; they are suitable for analyzing structures with axisymmetric geometry subjected to axisymmetric loading.

Abaqus/Standard also provides generalized plane strain elements, axisymmetric elements with twist, and axisymmetric elements with asymmetric deformation.

- Generalized plane strain elements include the additional generalization that the out-of-plane strain may vary linearly with position in the plane of the model. This formulation is particularly suited for the thermal-stress analysis of thick sections.
- Axisymmetric elements with twist model an initially axisymmetric geometry that can twist about the axis of symmetry. These elements are useful for modeling the torsion of cylindrical structures, such as axisymmetric rubber bushings.
- Axisymmetric elements with asymmetric deformation model an initially axisymmetric geometry that can deform asymmetrically (typically as a result of bending). They are useful for simulating problems such as an axisymmetric rubber mount that is subjected to shear loads.

The latter three classes of two-dimensional continuum elements are not discussed in this guide.

Two-dimensional solid elements must be defined in the 1–2 plane so that the node order is counterclockwise around the element perimeter, as shown in *Figure 2*.

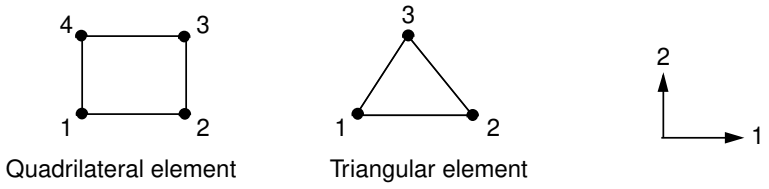


Figure 2: Correct nodal connectivity for two-dimensional elements.

When using a preprocessor to generate the mesh, ensure that the element normals all point in the same direction as the positive, global 3-axis. Failure to provide the correct element connectivity will cause Abaqus to issue an error message stating that elements have negative area.

Degrees of freedom

All of the stress/displacement continuum elements have translational degrees of freedom at each node. Correspondingly, degrees of freedom 1, 2, and 3 are active in three-dimensional elements, while only degrees of freedom 1 and 2 are active in plane strain elements, plane stress elements, and axisymmetric elements without twist. To find the active degrees of freedom in the other classes of two-dimensional solid elements, see *Two-Dimensional Solid Element Library*.

Element properties

All solid elements must refer to a solid section property that defines the material and any additional geometric data associated with the element. For three-dimensional and axisymmetric elements no additional geometric information is required: the nodal coordinates completely define the element geometry. For plane stress and plane strain elements the thickness of the elements may be specified or a default value of 1 will be used.

Formulation and integration

Alternative formulations available for the continuum family of elements in Abaqus/Standard include an *incompatible mode* formulation (the last or second-to-last letter in the element name is I) and a *hybrid* element formulation (the last letter in the element name is H), both of which are discussed in detail later in this guide.

In Abaqus/Standard you can choose between full and reduced integration for quadrilateral and hexahedral (brick) elements. In Abaqus/Explicit you can choose between full and reduced integration for hexahedral (brick) elements; however, only reduced integration is available for quadrilateral first-order elements. Both the formulation and type of integration can have a significant effect on the accuracy of solid elements, as discussed in *Element formulation and integration*.

Element output variables

By default, element output variables such as stress and strain refer to the global Cartesian coordinate system. Thus, the σ_{11}-component of stress at the integration point shown in *Figure 3*(a) acts in the global 1-direction. Even if the element rotates during a large-displacement simulation, as shown in *Figure 3*(b), the default is still to use the global Cartesian system as the basis for defining the element variables.

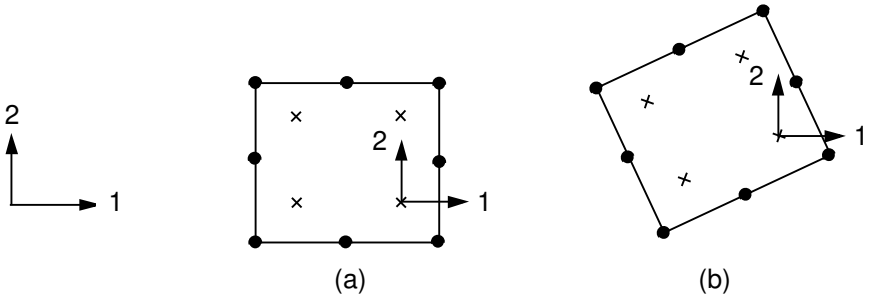


Figure 3: Default material directions for continuum elements.

However, Abaqus allows you to define a local coordinate system for element variables (see *Example: skew plate*). This local coordinate system rotates with the motion of the element in large-displacement simulations. A local coordinate system can be very useful if the object being modeled has some natural material orientation, such as the fiber directions in a composite material.

Shell elements

Shell elements are used to model structures in which one dimension (the thickness) is significantly smaller than the other dimensions and the stresses in the thickness direction are negligible.

Shell element names in Abaqus begin with the letter “S.” Axisymmetric shells all begin with the letters “SAX.” Abaqus/Standard also provides axisymmetric shells with asymmetric deformations, which begin with the letters “SAXA.” The first number in a shell element name indicates the number of nodes in the element, except for the case of axisymmetric shells, for which the first number indicates the order of interpolation.

Two types of shell elements are available in Abaqus: conventional shell elements and continuum shell elements. Conventional shell elements discretize a reference surface by defining the element's planar dimensions, its surface normal, and its initial curvature. Continuum shell elements, on the other hand, resemble three-dimensional solid elements in that they discretize an entire three-dimensional body yet are formulated so that their kinematic and constitutive behavior is similar to conventional shell elements. In this guide only conventional shell elements are discussed. Henceforth, we will refer to them simply as “shell elements.” For more information on continuum shell elements, see *About Shell Elements*.

The use of shell elements is discussed in detail in *Using Shell Elements*.

Shell element library

In Abaqus/Standard general three-dimensional shell elements are available with three different formulations: general-purpose, thin-only, and thick-only. The general-purpose shells and the axisymmetric shells with asymmetric deformation account for finite membrane strains and arbitrarily large rotations. The three-dimensional “thick” and “thin” element types provide for arbitrarily large rotations but only small strains. The general-purpose shells allow the shell thickness to change with the element deformation. All of the other shell elements assume small strains and no change in shell thickness, even though the element's nodes may undergo finite rotations. Triangular and quadrilateral elements with linear and quadratic interpolation are available. Both linear and quadratic axisymmetric shell elements are available. All of the quadrilateral shell elements (except for S4) and the triangular shell element S3/S3R use reduced integration. The S4 element and the other triangular shell elements use full integration. *Table 1* summarizes the shell elements available in Abaqus/Standard.

Table 1: Three classes of shell elements in Abaqus/Standard.

General-Purpose Shells	Thin-Only Shells	Thick-Only Shells
S4, S4R, S3/S3R, SAX1, SAX2, SAX2T, SC6R, SC8R	STRI3, STRI65, S4R5, S8R5, S9R5, SAXA	S8R, S8RT

All the shell elements in Abaqus/Explicit are general-purpose. Finite membrane strain and small membrane strain formulations are available. Triangular and quadrilateral elements are available with linear interpolation. A linear axisymmetric shell element is also available. *Table 2* summarizes the shell elements available in Abaqus/Explicit.

Table 2: Two classes of shell elements in Abaqus/Explicit.

Finite-Strain Shells	Small-Strain Shells
S4, S4R, S3/S3R, SAX1	S4RS, S4RSW, S3RS

For most explicit analyses the large-strain shell elements are appropriate. If, however, the analysis involves small membrane strains and arbitrarily large rotations, the small-strain shell elements are more computationally efficient. The S4RS and S3RS elements do not consider warping, while the S4RSW element does.

The shell formulations available in Abaqus are discussed in detail in *Using Shell Elements*.

Degrees of freedom

The three-dimensional elements in Abaqus/Standard whose names end in the number “5” (e.g., S4R5, STRI65) have 5 degrees of freedom at each node: three translations and two in-plane rotations (i.e., no rotations about the shell normal). However, all six degrees of freedom are activated at a node if required; for example, if rotational boundary conditions are applied or if the node is on a fold line of the shell.

The remaining three-dimensional shell elements have six degrees of freedom at each node (three translations and three rotations).

The axisymmetric shells have three degrees of freedom associated with each node:

1	Translation in the r-direction.
2	Translation in the z-direction.
6	Rotation in the r–z plane.

Element properties

All shell elements must refer to a shell section property that defines the thickness and material properties associated with the element.

The stiffness of the shell cross-section can be calculated either during the analysis or once at the beginning of the analysis.

If you choose to have the stiffness calculated during the analysis, Abaqus uses numerical integration to calculate the behavior at selected points through the thickness of the shell. These points are called section points, as shown in *Figure 1*. The associated material property definition may be linear or nonlinear. You can specify any odd number of section points through the shell thickness.

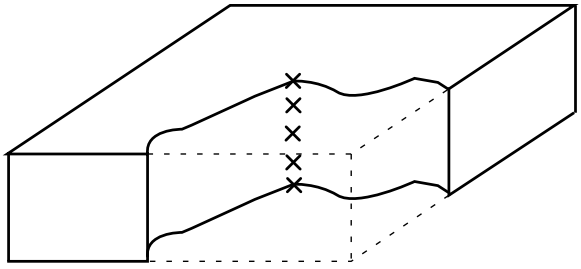


Figure 1: Section points through the thickness of a shell element.

If you choose to have the stiffness calculated once at the beginning of the analysis, you can define the cross-section behavior to model linear or nonlinear behavior. In this case Abaqus models the shell's cross-section behavior directly in terms of section engineering quantities (area, moments of inertia, etc.), so there is no need for Abaqus to integrate any quantities over the element cross-section. Therefore, this option is less expensive computationally. The response is calculated in terms of force and moment resultants; the stresses and strains are calculated only when they are requested for output. This approach is recommended when the response of the shell is linear elastic.

Element output variables

The element output variables for shells are defined in terms of local material directions that lie on the surface of each shell element. In all large-displacement simulations these axes rotate with the element's deformation. You can also define a local material coordinate system that rotates with the element's deformation in a large-displacement analysis.

Beam elements

Beam elements are used to model components in which one dimension (the length) is significantly greater than the other two dimensions and only the stress in the direction along the axis of the beam is significant.

Beam element names in Abaqus begin with the letter “B.” The next character indicates the dimensionality of the element: “2” for two-dimensional beams and “3” for three-dimensional beams. The third character indicates the interpolation used: “1” for linear interpolation, “2” for quadratic interpolation, and “3” for cubic interpolation.

The use of beam elements is discussed in *Using Beam Elements*.

Beam element library

Linear, quadratic, and cubic beams are available in two and three dimensions. Cubic beams are not available in Abaqus/Explicit.

Degrees of freedom

Three-dimensional beams have six degrees of freedom at each node: three translational degrees of freedom (1–3) and three rotational degrees of freedom (4–6). “Open-section”-type beams (such as B31OS) are available in Abaqus/Standard and have an additional degree of freedom (7) that represents the warping of the beam cross-section.

Two-dimensional beams have three degrees of freedom at each node: two translational degrees of freedom (1 and 2) and one rotational degree of freedom (6) about the normal to the plane of the model.

Element properties

All beam elements must refer to a beam section property that defines the material associated with the element as well as the beam section profile (i.e., the element's cross-sectional geometry); the nodal coordinates define only the length. You can define the beam section profile geometrically by specifying the shape and dimensions of the section. Alternatively, you can define a generalized beam section profile by specifying the section engineering properties, such as area and moment of inertia.

If you define the beam section profile geometrically, Abaqus calculates the cross-section behavior of the beam by numerical integration over the cross-section, allowing both linear and nonlinear material behavior.

If you provide the section engineering properties (area, moments of inertia, and torsional constants) instead of the cross-section dimensions, there is no need for Abaqus to integrate any quantities over the element cross-section. Therefore, this option is less expensive computationally. With this approach, the material behavior may be either linear or nonlinear. The response is calculated in terms of the force and moment resultants; the stresses and strains are calculated only when they are requested for output.

In Abaqus/Standard you can also define beams with linearly tapered cross-sections. General beam sections with linear response and standard library sections are supported.

Formulation and integration

The linear beams (B21 and B31) and the quadratic beams (B22 and B32) are shear deformable and account for finite axial strains; therefore, they are suitable for modeling both slender and stout beams. The cubic beam elements in Abaqus/Standard (B23 and B33) do not account for shear flexibility and assume small axial strain, although large displacements and rotations of the beams are valid. They are, therefore, suitable for modeling slender beams.

Abaqus/Standard provides variants of linear and quadratic beam elements that are suitable for modeling thin-walled, open-section beams (B31OS and B32OS). These elements model the effects of torsion and warping in open cross-sections, such as I-beams or U-section channels. Open-section beams are not covered in this guide.

Abaqus/Standard also has hybrid beam elements that are used for modeling very slender members, such as flexible risers on offshore oil installations, or for modeling very stiff links. Hybrid beams are not covered in this guide.

Element output variables

The stress components in three-dimensional, shear-deformable beam elements are the axial stress (σ_{11}) and the shear stress due to torsion (σ_{12}). The shear stress acts about the section wall in a thin-walled section. Corresponding strain measures are also available. The shear-deformable beams also provide estimates of transverse shear forces on the section. The slender (cubic) beams in Abaqus/Standard have only the axial variables as output. Open-section beams in space also have only the axial variables as output, since the torsional shear stresses are negligible in this case.

All two-dimensional beams use only axial stress and strain.

The axial force, bending moments, and curvatures about the local beam axes can also be requested for output. For details of what components are available with which elements, see *About Beam Modeling*. Details of how the local beam axes are defined are given in *Using Beam Elements*.

Truss elements

Truss elements are rods that can carry only tensile or compressive loads. They have no resistance to bending; therefore, they are useful for modeling pin-jointed frames. Moreover, truss elements can be used as an approximation for cables or strings (for example, in a tennis racket). Trusses are also sometimes used to represent reinforcement within other elements. The overhead hoist model in *Abaqus Basics* uses truss elements.

All truss element names begin with the letter "T." The next two characters indicate the dimensionality of the element—"2D" for two-dimensional trusses and "3D" for three-dimensional trusses. The final character represents the number of nodes in the element.

Truss element library

Linear and quadratic trusses are available in two and three dimensions. Quadratic trusses are not available in Abaqus/Explicit.

Degrees of freedom

Truss elements have only translational degrees of freedom at each node. Three-dimensional truss elements have degrees of freedom 1, 2, and 3, while two-dimensional truss elements have degrees of freedom 1 and 2.

Element properties

All truss elements must refer to a truss section property that associates a material property definition with the element and specifies its cross-sectional area.

Formulation and integration

In addition to the standard formulation, a hybrid truss element formulation is available in Abaqus/Standard. It is useful for modeling very stiff links whose stiffness is much greater than that of the overall structure.

Element output variables

Axial stress and strain are available as output for truss elements.

Rigid bodies

In Abaqus a rigid body is a collection of nodes and elements whose motion is governed by the motion of a single node, known as the *rigid body reference node*.

In this section:

- *Determining when to use a rigid body*
- *Components of a rigid body*
- *Rigid elements*

Determining when to use a rigid body

Rigid bodies can be used to model very stiff components that are either fixed or undergoing large rigid body motions. They can also be used to model constraints between deformable components, and they provide a convenient method of specifying certain contact interactions. When Abaqus is used for quasi-static forming analyses, rigid bodies are ideally suited for modeling tooling (such as a punch, die, drawbead, blank holder, roller, etc.) and may also be effective as a method of constraint.

Figure 1 shows a rigid body. The shape of the rigid body is defined either as an analytical surface obtained by revolving or extruding a two-dimensional geometric profile or as a discrete rigid body obtained by meshing the body with nodes and elements. The shape of the rigid body does not change during a simulation but can undergo large rigid body motions. The mass and inertia of a discrete rigid body can be calculated based on the contributions from its elements, or they can be assigned directly.

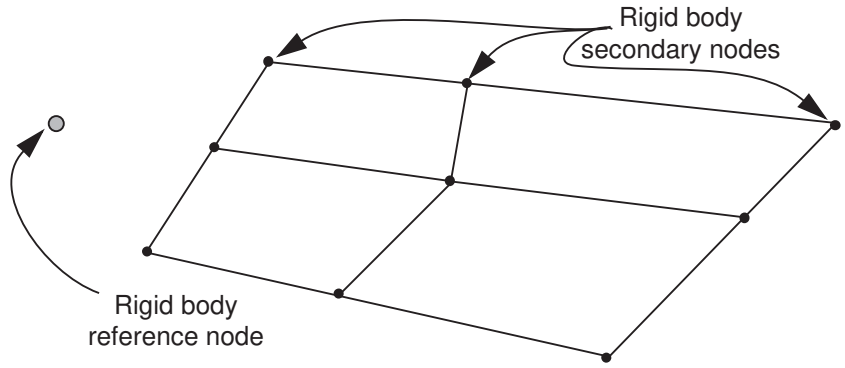


Figure 1: Elements forming a rigid body.

The motion of a rigid body can be prescribed by applying boundary conditions at the rigid body reference node. Loads on a rigid body are generated from concentrated loads applied to nodes and distributed loads applied to elements that are part of the rigid body or from loads applied to the rigid body reference node. Rigid bodies interact with the rest of the model through nodal connections to deformable elements and through contact with deformable elements.

The use of rigid bodies is illustrated in *Contact*.

It may be useful to make parts of a model rigid for verification purposes. For example, in complex models elements far away from the particular region of interest could be included as part of a rigid body, resulting in faster run times at the model development stage. When you are satisfied with the model, you can remove the rigid body definitions and incorporate an accurate deformable finite element representation throughout.

The principal advantage to representing portions of a model with rigid bodies rather than deformable finite elements is computational efficiency. Element-level calculations are not performed for elements that are part of a rigid body. Although some computational effort is required to update the motion of the nodes of the rigid body and to assemble concentrated and distributed loads, the motion of the rigid body is determined completely by a maximum of six degrees of freedom at the rigid body reference node.

In Abaqus/Explicit rigid bodies are particularly effective for modeling relatively stiff parts of a structure for which tracking stress waves and distributions is not important. Element stable time increment estimates in the stiff region can result in a very small global time increment. Since rigid bodies and elements that are part of a rigid body do not affect the global time increment, using a rigid body instead of a deformable finite element representation in a stiff region can result in a much larger global time increment, without significantly affecting the overall accuracy of the solution.

Rigid bodies defined with analytical rigid surfaces in Abaqus are slightly cheaper in terms of computational cost than discrete rigid bodies and may yield smoother results. In Abaqus/Explicit, for example, contact with analytical rigid surfaces tends to be less noisy than contact with discrete rigid bodies because analytical rigid surfaces can be smooth, whereas discrete rigid bodies are inherently faceted. However, the shapes that can be defined with analytical rigid surfaces are limited.

Components of a rigid body

The motion of a rigid body is controlled by the motion of a single node: the rigid body reference node. A rigid body reference node has both translational and rotational degrees of freedom and must be defined uniquely for every rigid body.

The position of the rigid body reference node is not important unless rotations are applied to the body or reaction moments about a certain axis through the body are desired. In either of these situations the node should be placed such that it lies on the desired axis through the body.

In addition to the rigid body reference node, discrete rigid bodies consist of a collection of nodes that are generated by assigning elements and nodes to the rigid body. These nodes, known as the *rigid body secondary nodes* (see *Rigid bodies*), provide a connection to other elements. Nodes that are part of a rigid body are one of two types:

- Pin nodes, which have only translational degrees of freedom.
- Tie nodes, which have both translational and rotational degrees of freedom.

The rigid body node type is determined by the type of elements on the rigid body to which the node is attached. The node type also can be specified or modified when assigning nodes directly to a rigid body. For pin nodes only the translational degrees of freedom are part of the rigid body, and the motion of these degrees of freedom is constrained by the motion of the rigid body reference node. For tie nodes both the translational and rotational degrees of freedom are part of the rigid body and are constrained by the motion of the rigid body reference node.

The nodes defining the rigid body cannot have any boundary conditions, multi-point constraints, or constraint equations applied to them. Boundary conditions, multi-point constraints, constraint equations, and loads can be applied, however, to the rigid body reference node.

Rigid elements

The rigid body capability in Abaqus allows most element types to be part of a rigid body. In addition, there exists an element type specifically designated as a "rigid element." The rules governing rigid bodies, such as how loads and boundary conditions are applied, pertain to all element types that form the rigid body, including rigid elements.

The names of all rigid elements begin with the letter "R." The next characters indicate the dimensionality of the element. For example, "2D" indicates that the element is planar; and "AX," that the element is axisymmetric. The final character represents the number of nodes in the element.

Rigid element library

The three-dimensional quadrilateral (R3D4) and triangular (R3D3) rigid elements are used to model the two-dimensional surfaces of a three-dimensional rigid body. Another element—a two-node, rigid beam element (RB3D2)—is provided in Abaqus/Standard mainly to model components of offshore structures to which fluid drag and buoyancy loads must be applied.

Two-node, rigid elements are available for plane strain, plane stress, and axisymmetric models. A planar, two-node rigid beam element is also available in Abaqus/Standard and is used mainly to model offshore structures in two dimensions.

Degrees of freedom

Only the rigid body reference node has independent degrees of freedom. For three-dimensional elements, the reference node has three translational and three rotational degrees of freedom; for planar and axisymmetric elements, the reference node has degrees of freedom 1, 2, and 6 (rotation about the 3-axis).

The nodes attached to rigid elements have only secondary degrees of freedom. The motion of these nodes is determined entirely by the motion of the rigid body reference node. For planar and three-dimensional rigid elements the only secondary degrees of freedom are translations. The rigid beam elements in Abaqus/Standard have the same secondary degrees of freedom as the corresponding deformable beam elements: 1–6 for the three-dimensional rigid beam and 1, 2, and 6 for the planar rigid beam.

Physical properties

All rigid elements must refer to a section property.

The following applies only to Abaqus/Explicit: For the planar and rigid beam elements the cross-sectional area can be defined. For the axisymmetric and three-dimensional elements the thickness can be defined. The default thickness is zero. These data are required only if you apply body forces to the rigid elements or when the thickness is needed for the contact definition.

Formulation and integration

Since the rigid elements are not deformable, they do not use numerical integration points, and there are no optional formulations.

Element output variables

There are no element output variables. The only output from rigid elements is the motion of the nodes. In addition, reaction forces and reaction moments are available at the rigid body reference node.

Summary

Abaqus has an extensive library of elements that can be used for a wide range of structural applications.

- Your choice of element type has important consequences regarding the accuracy and efficiency of your simulation. The elements available in Abaqus/Explicit are (in general) a subset of those available in Abaqus/Standard.
- The degrees of freedom active at a node depend on the element types attached to the node.
- The element name completely identifies the element's family, formulation, number of nodes, and type of integration.
- All elements must refer to a section property definition. The section property provides any additional data required to define the geometry of the element and also identifies the associated material property definition.
- For continuum elements Abaqus defines the element output variables, such as stress and strain, with respect to the global Cartesian coordinate system. You can change to a local material coordinate system.
- For three-dimensional shell elements Abaqus defines the element output variables with respect to a coordinate system based on the surface of the shell. You can define a local material coordinate system.
- For computational efficiency any part of a model can be defined as a rigid body, which has degrees of freedom only at its reference node.
- As a method of constraint in an Abaqus/Explicit analysis, rigid bodies are computationally more efficient than multi-point constraints.

Using Continuum Elements

The continuum (solid) family of stress/displacement elements is the most comprehensive of the element libraries in Abaqus.
There are some differences in the solid element libraries available in Abaqus/Standard and Abaqus/Explicit.

Abaqus/Standard solid element library

The Abaqus/Standard solid element library includes first-order (linear) interpolation elements and second-order (quadratic) interpolation elements in two or three dimensions using either full or reduced integration. Triangles and quadrilaterals are available in two dimensions; and tetrahedra, pyramids, triangular wedges, and hexahedra ("bricks") are provided in three dimensions. Modified second-order triangular and tetrahedral elements are also provided.

In addition, hybrid and incompatible mode elements are available in Abaqus/Standard.

Abaqus/Explicit solid element library

The Abaqus/Explicit solid element library includes reduced-integration first-order (linear) interpolation elements in two or three dimensions, second-order interpolation tetrahedra, and modified second-order interpolation triangles and tetrahedra. Full integration or regular second-order elements are not available in Abaqus/Explicit, with the exception of the fully integrated first-order hexahedral element (an incompatible mode version of this element is also available).

For detailed information on the options available for continuum elements, please see *Solid (Continuum) Elements*.

When the permutations of all these various element options are made, the total number of solid elements available to you is large—over 20 just for three-dimensional models. The accuracy of your simulation will depend strongly on the type of element you use in your model. The thought of choosing which of these elements is best for your model may seem daunting, especially at first. However, you will come to view this selection as a 20+ piece tool set that provides you with the ability to choose just the right tool, or element, for a particular job.

This chapter discusses the effect that different element formulations and levels of integration have on the accuracy of a particular analysis. Some general guidelines for selecting continuum elements are also given. These provide the foundation upon which you can build your knowledge as you gain more experience using Abaqus. The example at the end of this section will allow you to put this knowledge to use as you build and analyze a connecting lug.

In this section:

- *Element formulation and integration*
- *Selecting continuum elements*
- *Example: connecting lug*
- *Mesh convergence*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Element formulation and integration

The influence that the order of the element (linear or quadratic), the element formulation, and the level of integration have on the accuracy of a structural simulation will be demonstrated by considering a static analysis of the cantilever beam.

In this section:

- *About element formulation and integration*
- *Full integration*
- *Reduced integration*
- *Incompatible mode elements*
- *Hybrid elements*

About element formulation and integration

This is a classic test used to assess the behavior of a given finite element.

The cantilever beam is shown in *Figure 1*. Since the beam is relatively slender, we would normally model it with beam elements. However, it is used here to help assess the effectiveness of various solid elements.

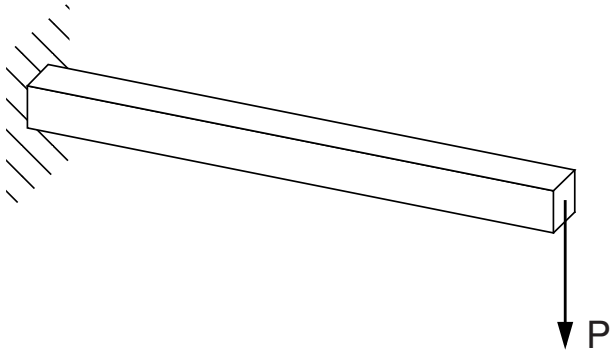


Figure 1: Cantilever beam under a point load *P* at its free end.

The beam is 150 mm long, 2.5 mm wide, and 5 mm deep; built-in at one end; and carrying a tip load of 5 N at the free end. The material has a Young's modulus, *E*, of 70 GPa and a Poisson's ratio of 0.0. Using beam theory, the static deflection of the tip of the beam for a load *P* is given as

$$\delta_{tip} = \frac{Pl^3}{3EI},$$

where $I = bd^3/12$, *l* is the length, *b* is the width, and *d* is the depth of the beam.

For $P =$ 5 N the tip deflection is 3.09 mm.

Full integration

The expression “full integration” refers to the number of Gauss points required to integrate the polynomial terms in an element's stiffness matrix exactly when the element has a regular shape. For hexahedral and quadrilateral elements a “regular shape” means that the edges are straight and meet at right angles and that any edge nodes are at the midpoint of the edge. Fully integrated, linear elements use two integration points in each direction. Thus, the three-dimensional element C3D8 uses a $2 \times 2 \times 2$ array of integration points in the element. Fully integrated, quadratic elements (available only in Abaqus/Standard) use three integration points in each direction. The locations of the integration points in fully integrated, two-dimensional, quadrilateral elements are shown in *Figure 1*.

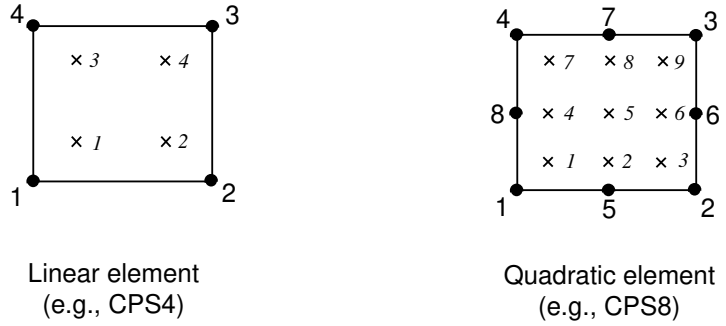


Figure 1: Integration points in fully integrated, two-dimensional, quadrilateral elements.

Several different finite element meshes were used in Abaqus/Standard simulations of the cantilever beam problem, as shown in *Figure 2*. The simulations use either linear or quadratic, fully integrated elements and illustrate the effects of both the order of the element (first versus second) and the mesh density on the accuracy of the results.
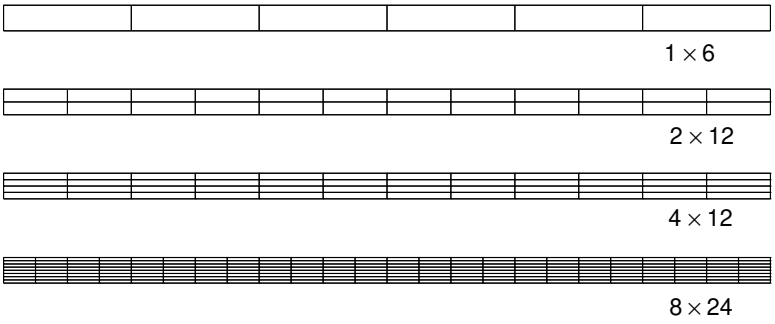


Figure 2: Meshes used for the cantilever beam simulations.

The ratios of the tip displacements for the various simulations to the beam-theory value of 3.09 mm are shown in *Table 1*.

Table 1: Normalized tip displacements with fully-integrated elements.

Element	Mesh Size (Depth × Length)			
	1 × 6	2 × 12	4 × 12	8 × 24
CPS4	0.074	0.242	0.242	0.561
CPS8	0.994	1.000	1.000	1.000
C3D8	0.077	0.248	0.243	0.563
C3D20	0.994	1.000	1.000	1.000

The linear elements CPS4 and C3D8 underpredict the deflection so badly that the results are unusable. The results are least accurate with coarse meshes, but even a fine mesh (8 × 24) still predicts a tip displacement that is only 56% of the theoretical value. Notice that for the linear, fully integrated elements it makes no difference how many elements there are through the thickness of the beam. The underprediction of tip deflection is caused by *shear locking*, which is a problem with all fully integrated, first-order, solid elements.

As we have seen, shear locking causes the elements to be too stiff in bending. It is explained as follows. Consider a small piece of material in a structure subject to pure bending. The material will distort as shown in *Figure 3*.

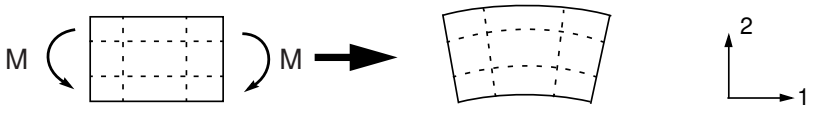


Figure 3: Deformation of material subjected to bending moment *M*.

Lines initially parallel to the horizontal axis take on constant curvature, and lines through the thickness remain straight. The angle between the horizontal and vertical lines remains at 90°.

The edges of a linear element are unable to curve; therefore, if the small piece of material is modeled using a single element, its deformed shape is like that shown in *Figure 4*.



Figure 4: Deformation of a fully integrated, linear element subjected to bending moment *M*.

For visualization, dotted lines that pass through the integration points are plotted. It is apparent that the upper line has increased in length, indicating that the direct stress in the 1-direction, σ_{11}, is tensile. Similarly, the length of the lower dotted line has decreased, indicating that σ_{11} is compressive. The length of the vertical dotted lines has not changed (assuming that displacements are small); therefore, σ_{22} at all integration points is zero. All this is consistent with the expected state of stress of a small piece of material subjected to pure bending. However, at each integration point the angle between the vertical and horizontal lines, which was initially 90°, has changed. This indicates that the shear stress, σ_{12}, at these points is nonzero. This is incorrect: the shear stress in a piece of material under pure bending is zero.

This spurious shear stress arises because the edges of the element are unable to curve. Its presence means that strain energy is creating shearing deformation rather than the intended bending deformation, so the overall deflections are less: the element is too stiff.

Shear locking only affects the performance of fully integrated, linear elements subjected to bending loads. These elements function perfectly well under direct or shear loads. Shear locking is not a problem for quadratic elements since their edges are able to curve (see *Figure 5*). The predicted tip displacements for the quadratic elements shown in *Table 1* are close to the theoretical value. However, quadratic elements will also exhibit some locking if they are distorted or if the bending stress has a gradient, both of which can occur in practical problems.

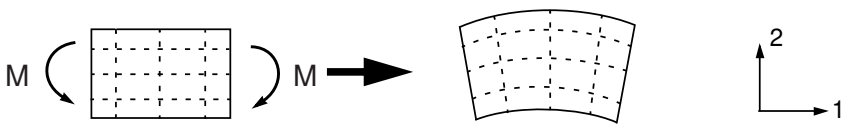


Figure 5: Deformation of a fully integrated, quadratic element subjected to bending moment *M*.

Fully integrated, linear elements should be used only when you are fairly certain that the loads will produce minimal bending in your model. Use a different element type if you have doubts about the type of deformation the loading will

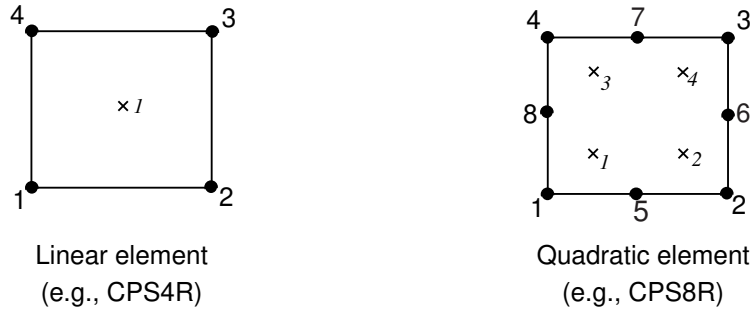
create. Fully integrated, quadratic elements can also lock under complex states of stress; thus, you should check the results carefully if they are used exclusively in your model. However, they are very useful for modeling areas where there are local stress concentrations.

Volumetric locking is another form of overconstraint that occurs in fully integrated elements when the material behavior is (almost) incompressible. It causes overly stiff behavior for deformations that should cause no volume changes. It is discussed further in *Materials*.

Reduced integration

Only quadrilateral and hexahedral elements can use a reduced-integration scheme; all wedge, pyramid, tetrahedral, and triangular solid elements use full integration, although they can be used in the same mesh with reduced-integration hexahedral or quadrilateral elements.

Reduced-integration elements use one fewer integration point in each direction than the fully integrated elements. Reduced-integration, linear elements have just a single integration point located at the element's centroid. (Actually, these first-order elements in Abaqus use the more accurate “uniform strain” formulation, where average values of the strain components are computed for the element. This distinction is not important for this discussion.) The locations of the integration points for reduced-integration, quadrilateral elements are shown in *Figure 1*.



Linear element
(e.g., CPS4R)

Quadratic element
(e.g., CPS8R)

Figure 1: Integration points in two-dimensional elements with reduced integration.

Abaqus simulations of the cantilever beam problem were performed using the reduced-integration versions of the same four elements utilized previously and using the four finite element meshes shown in *Figure 2*. The results from these simulations are presented in *Table 1*.

Table 1: Normalized tip displacements with reduced-integration elements.

Element	Mesh Size (Depth × Length)			
	1 × 6	2 × 12	4 × 12	8 × 24
CPS4R	20.3*	1.308	1.051	1.012
CPS8R	1.000	1.000	1.000	1.000
C3D8R	70.1*	1.323	1.063	1.015
C3D20R	0.999**	1.000	1.000	1.000
* no stiffness to resist the applied load, ** two elements through width				

Linear reduced-integration elements tend to be too flexible because they suffer from their own numerical problem called *hourglassing*. Again, consider a single reduced-integration element modeling a small piece of material subjected to pure bending (see *Figure 2*).

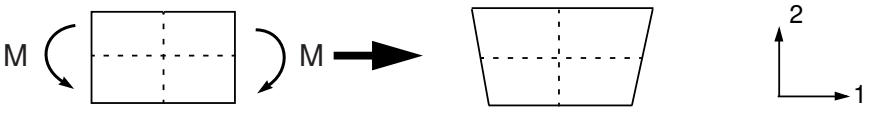


Figure 2: Deformation of a linear element with reduced integration subjected to bending moment *M*.

Neither of the dotted visualization lines has changed in length, and the angle between them is also unchanged, which means that all components of stress at the element's single integration point are zero. This bending mode of deformation is thus a zero-energy mode because no strain energy is generated by this element distortion. The element is unable to resist this type of deformation since it has no stiffness in this mode. In coarse meshes this zero-energy mode can propagate through the mesh, producing meaningless results.

In Abaqus a small amount of artificial “hourglass stiffness” is introduced in first-order reduced-integration elements to limit the propagation of hourglass modes. This stiffness is more effective at limiting the hourglass modes when more elements are used in the model, which means that linear reduced-integration elements can give acceptable results as long as a reasonably fine mesh is used. The errors seen with the finer meshes of linear reduced-integration elements (see *Table 1*) are within an acceptable range for many applications. The results suggest that at least four elements should be used through the thickness when modeling any structures carrying bending loads with this type of element. When a single linear reduced-integration element is used through the thickness of the beam, all the integration points lie on the neutral axis and the model is unable to resist bending loads. (These cases are marked with a * in *Table 1*.)

Linear reduced-integration elements are very tolerant of distortion; therefore, use a fine mesh of these elements in any simulation where the distortion levels may be very high.

The quadratic reduced-integration elements available in Abaqus/Standard also have hourglass modes. However, the modes are almost impossible to propagate in a normal mesh and are rarely a problem if the mesh is sufficiently fine. The 1×6 mesh of C3D20R elements fails to converge because of hourglassing unless two elements are used through the width, but the more refined meshes do not fail even when only one element is used through the width. Quadratic reduced-integration elements are not susceptible to locking, even when subjected to complicated states of stress. Therefore, these elements are generally the best choice for most general stress/displacement simulations, except in large-displacement simulations involving very large strains and in some types of contact analyses.

Incompatible mode elements

The incompatible mode elements, available primarily in Abaqus/Standard, are an attempt to overcome the problems of shear locking in fully integrated, first-order elements. Since shear locking is caused by the inability of the element's displacement field to model the kinematics associated with bending, additional degrees of freedom, which enhance the element's deformation gradient, are introduced into the first-order element. These enhancements to the deformation gradient allow a first-order element to have a linear variation of the deformation gradient across the element's domain as shown in *Figure 1*(a). The standard element formulation results in a constant deformation gradient across the element as shown in *Figure 1*(b), resulting in the nonzero shear stress associated with shear locking.

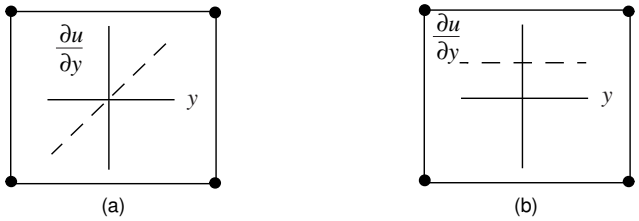


Figure 1: Variation of deformation gradient in (a) an incompatible mode (enhanced deformation gradient) element and (b) a first-order element using a standard formulation.

These enhancements to the deformation gradient are entirely internal to an element and are not associated with nodes positioned along the element edges. Unlike incompatible mode formulations that enhance the displacement field directly, the formulation used in Abaqus does not result in overlapping material or a hole along the boundary between two elements, as shown in *Figure 2*.

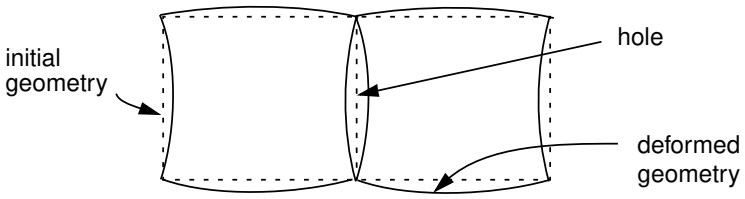


Figure 2: Potential kinematic incompatibility between incompatible mode elements that use enhanced displacement fields rather than enhanced deformation gradients. Abaqus uses the latter formulation for its incompatible mode elements.

Furthermore, the formulation used in Abaqus is extended easily to nonlinear, finite-strain simulations, something which is not as easy with the enhanced displacement field elements.

Incompatible mode elements can produce results in bending problems that are comparable to quadratic elements but at significantly lower computational cost. However, they are sensitive to element distortions. *Figure 3* shows the cantilever beam modeled with deliberately distorted incompatible mode elements: in one case with “parallel” distortion and in the other with “trapezoidal” distortion.

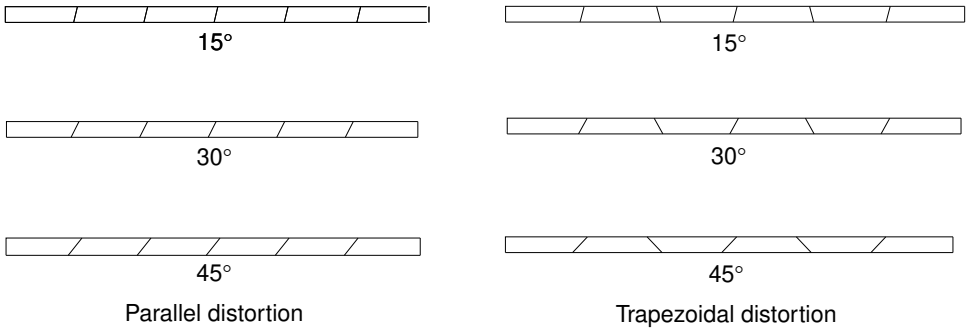


Figure 3: Distorted meshes of incompatible mode elements.

Figure 4 shows the tip displacements for the cantilever beam models. The tip displacements are normalized with respect to the analytical solution and plotted against the level of element distortion.

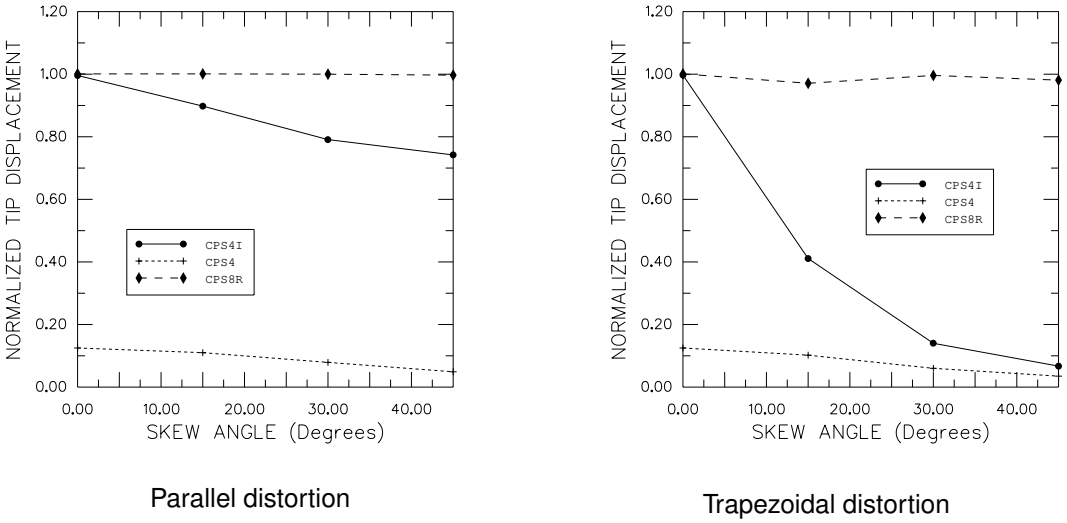


Figure 4: Effect of parallel and trapezoidal distortion of incompatible mode elements.

Three types of plane stress elements in Abaqus/Standard are compared: the fully integrated, linear element; the reduced-integration, quadratic element; and the linear, incompatible mode element. The fully integrated, linear elements produce poor results in all cases, as expected. On the other hand, the reduced-integration, quadratic elements give very good results that do not deteriorate until the elements are badly distorted.

When the incompatible mode elements are rectangular, even a mesh with just one element through the thickness of the cantilever gives results that are very close to the theoretical value. However, even quite small levels of trapezoidal distortion make the elements much too stiff. Parallel distortion also reduces the accuracy of the element but to a lesser extent.

Incompatible mode elements are useful because they can provide high accuracy at a low cost if they are used appropriately. However, care must be taken to ensure that the element distortions are small, which may be difficult when meshing complex geometries; therefore, you should again consider using the reduced-integration, quadratic elements in models with such geometries because they show much less sensitivity to mesh distortion. In a severely distorted mesh, however, simply changing the element type will generally not produce accurate results. The mesh distortion should be minimized as much as possible to improve the accuracy of the results.

Hybrid elements

A hybrid element formulation is available for just about every type of continuum element in Abaqus/Standard, including all reduced-integration and incompatible mode elements. Hybrid elements are not available in Abaqus/Explicit with the exception of the linear hybrid tetrahedron element. Elements using this formulation have the letter “H” in their names.

Hybrid elements are used when the material behavior is incompressible (Poisson's ratio = 0.5) or very close to incompressible (Poisson's ratio > 0.475). Rubber is an example of a material with incompressible material behavior. An incompressible material response cannot be modeled with regular elements (except in the case of plane stress) because the pressure stress in the element is indeterminate. Consider an element under uniform hydrostatic pressure (*Figure 1*).

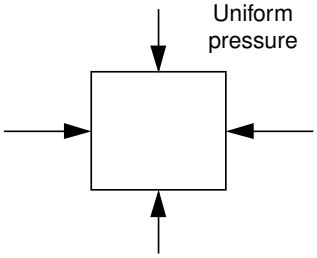


Figure 1: Element under hydrostatic pressure.

If the material is incompressible, its volume cannot change under this loading. Therefore, the pressure stress cannot be computed from the displacements of the nodes; and, thus, a pure displacement formulation is inadequate for any element with incompressible material behavior.

Hybrid elements include an additional degree of freedom that determines the pressure stress in the element directly. The nodal displacements are used only to calculate the deviatoric (shear) strains and stresses.

A more detailed description of the analysis of rubber materials is given in *Materials*.

Selecting continuum elements

The correct choice of element for a particular simulation is vital if accurate results are to be obtained at a reasonable cost. You will undoubtedly develop your own guidelines for selecting elements for your own particular applications as you become more experienced in using Abaqus. However, as you begin to use Abaqus, the guidelines given here may be helpful.

The following recommendations apply to both Abaqus/Standard and Abaqus/Explicit:

- Minimize the mesh distortion as much as possible. Coarse meshes with distorted linear elements can give very poor results.
- Use a fine mesh of linear, reduced-integration elements (CAX4R, CPE4R, CPS4R, C3D8R, etc.) for simulations involving very large mesh distortions (large-strain analysis).
- In three dimensions use hexahedral (brick-shaped) elements wherever possible. They give the best results for the minimum cost. Complex geometries can be difficult to mesh completely with hexahedrons; therefore, wedge, pyramid, and tetrahedral elements may be necessary. The linear versions of these elements, C3D4, C3D5, and C3D6, are poor elements (fine meshes are needed to obtain accurate results); as a result, these elements should generally be used only when necessary to complete a mesh, and, even then, they should be far from any areas where accurate results are needed.
- Some preprocessors contain free-meshing algorithms that mesh arbitrary geometries with tetrahedral elements. The quadratic tetrahedral elements in Abaqus/Standard (C3D10 or C3D10HS) are suitable for general usage; but when used with contact, they should be used only with the “surface-to-surface” contact discretization. An alternative to these elements is the modified quadratic tetrahedral element (C3D10M) available in both analysis products. This element is robust for large-deformation problems and contact problems using either the traditional “node-to-surface” or the “surface-to-surface” contact discretization and exhibits minimal shear and volumetric locking. With either type of element, however, the analysis will take longer to run than an equivalent mesh of hexahedral elements. You should not use a mesh containing only linear tetrahedral elements (C3D4): the results will be inaccurate unless you use an extremely large number of elements.

Abaqus/Standard users should also consider the following recommendations:

- Use quadratic, reduced-integration elements (CAX8R, CPE8R, CPS8R, C3D20R, etc.) for general analysis work, unless you need to model very large strains or have a simulation with complex, changing contact conditions.
- Use quadratic, fully integrated elements (CAX8, CPE8, CPS8, C3D20, etc.) locally where stress concentrations may exist. They provide the best resolution of the stress gradients at the lowest cost.
- For contact problems use a fine mesh of linear, reduced-integration elements or incompatible mode elements (CAX4I, CPE4I, CPS4I, C3D8I, etc.). See *Contact*.

Example: connecting lug

In this example you will use three-dimensional, continuum elements to model a connecting lug.

In this section:

- *Connecting lug analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing—visualizing the results*
- *Rerunning the analysis using Abaqus/Explicit*
- *Postprocessing the dynamic analysis results*

Connecting lug analysis

The lug is welded firmly to a massive structure at one end. The other end contains a hole. When it is in service, a bolt will be placed through the hole of the lug.

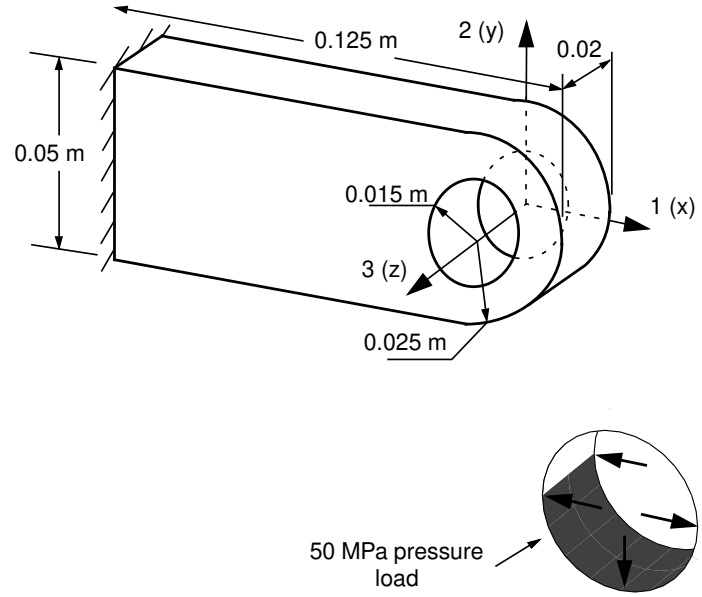


Figure 1: Sketch of the connecting lug.

You have been asked to determine the static deflection of the lug when a 30 kN load is applied to the bolt in the negative 2-direction. Because the goal of this analysis is to examine the static response of the lug, you should use Abaqus/Standard as your analysis product. You decide to simplify this problem by making the following assumptions:

- Rather than include the complex bolt-lug interaction in the model, you will use a distributed pressure over the bottom half of the hole to load the connecting lug (see *Figure 1*).
- You will neglect the variation of pressure magnitude around the circumference of the hole and use a uniform pressure.
- The magnitude of the applied uniform pressure will be 50 MPa: 30 kN/ (2 × 0.015 m × 0.02 m).

After examining the static response of the lug, you will modify the model and use Abaqus/Explicit to study the transient dynamic effects resulting from sudden loading of the lug.

Preprocessing—creating the model with Abaqus/CAE

In this section we discuss how to use Abaqus/CAE to create the entire model for this simulation.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Connecting lug*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Connecting lug**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

In this section:

- *Starting Abaqus/CAE*
- *Defining the model geometry*
- *Defining the material and section properties*
- *Creating an assembly*
- *Defining steps and specifying output requests*
- *Prescribing boundary conditions and applied loads*
- *Designing the mesh: partitioning and creating the mesh*
- *Creating, running, and monitoring a job*

Starting Abaqus/CAE

Start Abaqus/CAE by typing

```
abaqus cae
```

at the operating system prompt, where `abaqus` is the command used to run Abaqus on your system. From the **Create Model Database** options in the **Start Session** dialog box that appears, select **With Standard/Explicit Model**.

Defining the model geometry

As always, the first step in creating the model is to define its geometry. In this example you will create a three-dimensional, deformable body with a solid, extruded base feature. You will first sketch the two-dimensional profile of the lug and then extrude it.

You need to decide what system of units to use in your model. The SI system of meters, seconds, and kilograms is recommended; but you can use another system if you prefer.

1. In the Model Tree, double-click the **Parts** container to create a new part.
2. In the **Create Part** dialog box, name the part `Lug` and accept the default settings of a three-dimensional, deformable body and a solid, extruded base feature. In the **Approximate size** text field, type `0.250`. This value is twice the largest dimension of the part. Click **Continue** to exit the **Create Part** dialog box.
3. Use the dimensions given in *Figure 1* to sketch the profile of the lug. The following possible approach can be used:
   a. Create an arbitrary rectangle using the **Create Lines: Rectangle** tool. Delete the right vertical edge, and assign an **Equal length** constraint to the top and bottom edges using the constraints tool . Use the dimensions tool  to adjust the profile so that it is 0.100 m long × 0.050 m wide, as shown in *Figure 1*.

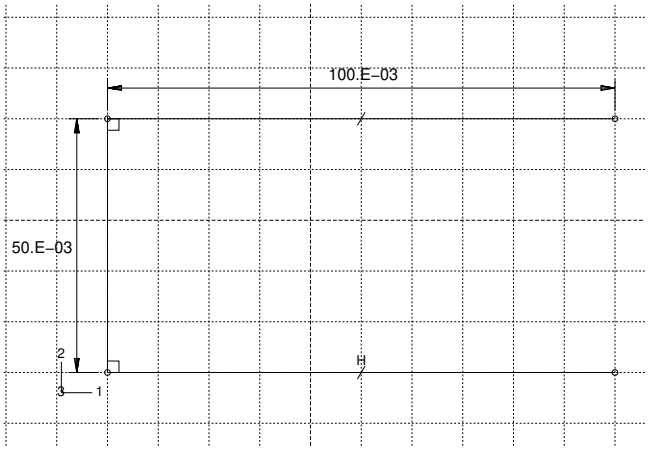


Figure 1: Open rectangle (with grid spacing doubled).



Note:

The figures in this section include dimensions and constraints for the purpose of controlling the shape of the sketch. As seen previously, these tools are accessible from the Sketcher toolbox. These tools are also accessible by selecting **Add->Constraint** and **Add->Dimension**, respectively, from the main menu bar.

You can edit the dimension values as you add them to the sketch, or you can edit the value of existing dimensions by selecting **Edit->Dimension** from the main menu bar or by using the **Edit Dimension Value**  tool.

b. Close the profile by adding a semicircular arc, as shown in *Figure 2*, using the **Create Arc: Thru 3 points** tool, .

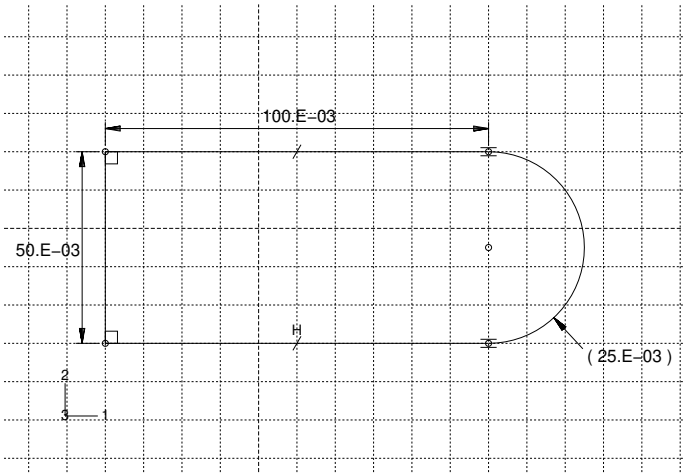


Figure 2: Rounded end (with grid spacing doubled).

Select the two vertices at the open end of the rectangle as the endpoints of the arc, starting with the top one. Select any point to the right of the sketch as a point that lies on the arc. Define **Tangent** constraints between the ends of the arc and the horizontal lines to refine the sketch. In *Figure 2* the radial dimension of the arc is enclosed in parentheses, indicating that it is used only for reference purposes. To change any dimension to a reference value, simply edit the dimension and toggle on **Reference** in the **Edit Dimension** dialog box.

c. Sketch a circle of radius 0.015 m, as shown in *Figure 3*, using the **Create Circle: Center and Perimeter** tool, .

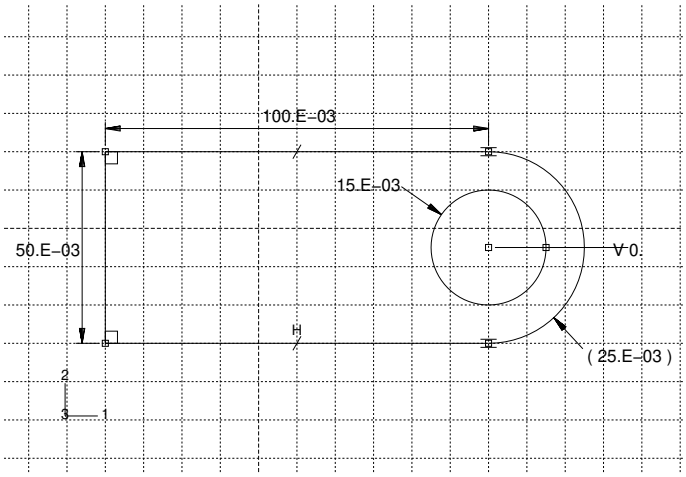


Figure 3: Lug hole (with grid spacing doubled).

Place the center of the circle to coincide approximately with the center of the arc created in the previous step. The perimeter point should be placed to the right of the center point. Apply a

Concentric constraint to the arc and the circle. Use the **Add Dimension**  tool to change the value of the radius to 0.015 m.

d. The perimeter points of the circles should be on the same horizontal plane as the circle center points (as in *Figure 3*). If these points do not line up correctly, dimension the vertical distance between the center of each circle and its perimeter point so that the distance is 0.



Note:

When you mesh a part, Abaqus/CAE places nodes wherever vertices appear along an edge; therefore, the location of the vertex on the circumference of the circle influences the final mesh. Placing it on the same horizontal plane as the center point results in a higher quality mesh.

e. Click **Done** in the prompt area when you are finished sketching the profile.

The **Edit Base Extrusion** dialog box appears. To complete the part definition, you must specify the distance over which the profile will be extruded.

f. In the dialog box, enter an extrusion distance of `0.020` m and click **OK**.

Abaqus/CAE exits the Sketcher and displays the part.

Defining the material and section properties

The next step in creating the model involves defining and assigning material and section properties to the part. Each region of a deformable body must refer to a section property, which includes the material definition. In this model you will create a single linear elastic material with a Young's modulus E = 200 GPa and Poisson's ratio ν = 0.3.

Define material properties

1. In the Model Tree, double-click the **Materials** container to create a new material definition.
2. In the material editor that appears, name the material `Steel` and select **Mechanical->Elasticity->Elastic**. Enter `200.0E9` for the **Young's Modulus** and `0.3` for the **Poisson's Ratio**. Click **OK**.

Define section properties

1. In the Model Tree, double-click the **Sections** container to create a new section definition. Accept the default solid, homogeneous section type; and name the section `LugSection`. Click **Continue**.
2. In the **Edit Section** dialog box that appears, accept **Steel** as the material and click **OK**.

Assign section properties

1. In the Model Tree, expand the **Lug** item underneath the **Parts** container and double-click **Section Assignments** in the list of part attributes that appears.
2. Select the entire part as the region to which the section will be assigned by clicking on it. When the part is highlighted, click **Done** in the prompt area.
3. In the **Edit Section Assignment** dialog box that appears, accept **LugSection** as the section definition, and click **OK**.

Creating an assembly

An assembly contains all the geometry included in the finite element model. Each Abaqus/CAE model contains a single assembly. The assembly is initially empty, even though you have already created a part. You will create an instance of the part in the Assembly module to include it in your model.

1. In the Model Tree, expand the **Assembly** container and double-click **Instances** in the list that appears to create an instance of the part.
2. In the **Create Instance** dialog box, select **Lug** from the **Parts** list and click **OK**.

   The model is oriented by default so that the global 1-axis lies along the length of the lug, the global 2-axis is vertical, and the global 3-axis lies in the thickness direction.

Defining steps and specifying output requests

You will now define the analysis steps.
Since interactions, loads, and boundary conditions can be step dependent, analysis steps must be defined before these can be specified. For this simulation you will define a single static, general step. In addition, you will specify output requests for your analysis. These requests will include output to the output database (`.odb`) file.

Define a step

1. In the Model Tree, double-click the **Steps** container to create an analysis step. In the **Create Step** dialog box that appears, name the step `LugLoad` and accept the **General** procedure type. From the list of available procedure options, accept **Static, General**. Click **Continue**.
2. In the **Edit Step** dialog box that appears, enter the following step description: `Apply uniform pressure to the hole`. Accept the default settings, and click **OK**.

Specify output requests to the `.odb` file

Since you will use the Visualization module to postprocess the results, you must specify the output data you wish to have written to the output database file. Default history and field output requests are selected automatically by Abaqus/CAE for each procedure type. Edit these requests so that only the displacements, stresses, and reaction forces are written as field data to the output database file.

1. In the Model Tree, click mouse button 3 on the **Field Output Requests** container and select **Manager** from the menu that appears.
2. In the **Field Output Requests Manager** that appears, select the cell labeled `Created` in the column labeled **LugLoad** if it is not already selected. The information at the bottom of the dialog box indicates that preselected default field output requests have been made for this step.
3. On the right side of the dialog box, click **Edit** to change the field output requests. In the **Edit Field Output Request** dialog box that appears:
   a. Click the arrow next to **Stresses** to show the list of available stress output. Accept the default selection of stress components and invariants.
   b. Under **Forces/Reactions**, make the following changes:
      1. Toggle off concentrated force and moment output (CF).
      2. Toggle on nodal forces due to element stresses (NFORC).
   c. Toggle off **Strains** and **Contact**.
   d. Accept the default **Displacement/Velocity/Acceleration** output.
   e. Click **OK**, and click **Dismiss** to close the **Field Output Requests Manager**.

   Note: The option **Exterior only** in the **Edit Field Output Request** dialog box can be used to restrict field output to nodes and elements that belong to the exterior of three-dimensional elements in a model; this option reduces the size of the output database file.

4. Delete all history output requests. In the Model Tree, click mouse button 3 on the **History Output Requests** container and select **Manager** to open the **History Output Requests Manager**. In the **History Output Requests Manager**, select the cell labeled `Created` in the column labeled **LugLoad** if it is not already selected. At the bottom of the dialog box, click **Delete** and click **Yes** in the warning dialog box that appears. Click **Dismiss** to close the **History Output Requests Manager**.

Prescribing boundary conditions and applied loads

In this model the left-hand end of the connecting lug needs to be constrained in all three directions.
This region is where the lug is attached to its parent structure (see *Figure 1*). In Abaqus/CAE boundary conditions are applied to geometric regions of a part rather than to the finite element mesh itself. This association between boundary conditions and part geometry makes it very easy to vary the mesh without having to respecify the boundary conditions. The same holds true for load definitions.

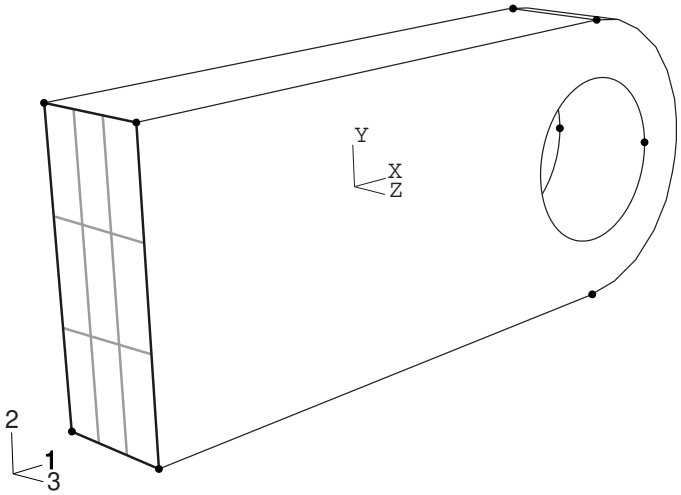


Figure 1: Built-in end of the connecting lug.

The lug carries a pressure of 50 MPa distributed around the bottom half of the hole. To apply the load correctly, however, the part must first be partitioned (i.e., divided) so that the hole is composed of two regions: a top half and a bottom half.

You use the Partition toolset to divide a part or assembly into regions. Partitioning is used for many reasons; it is commonly used for the purposes of defining material boundaries, indicating the location of loads and constraints (as in this example), and refining the mesh. An example of the use of partitioning for meshing purposes is discussed in the next section. For more information on partitioning, see *The Partition toolset*.

Dependent part instances cannot be modified at the assembly level (e.g., they cannot be partitioned in an assembly-level module). The reason for this restriction is that all dependent instances of a part must have identical geometry so they can share the same mesh topology as the original part. Thus, any change to a dependent part instance has to be made to the original part itself (i.e., at the part level). In contrast, independent part instances may be partitioned at the assembly level. In this example a dependent part instance (the default) was created; the corresponding partitioning instructions follow.

Prescribe boundary conditions

1. In the Model Tree, double-click the **BCs** container to prescribe boundary conditions on the model. In the **Create Boundary Condition** dialog box that appears, name the boundary condition `Fix left end`, and select `LugLoad` as the step in which it will be applied (since it is a fixed condition, it can be applied either in the initial step or the analysis step; here we choose the analysis step for convenience). Accept **Mechanical** as the category and `Symmetry/Antisymmetry/Encastre` as the type. Click **Continue**.
2. You may need to rotate the view to facilitate your selection in the following steps. Select **View->Rotate** from the main menu bar (or use the  tool from the View Manipulation toolbar) and drag the cursor over the virtual trackball in the viewport. The view rotates interactively; try dragging the cursor inside

and outside the virtual trackball to see the difference in behavior. Click mouse button 2 to exit the rotate view tool before proceeding.

3. Select the left end of the lug (indicated in *Figure 1*) using the cursor. Click **Done** in the prompt area when the appropriate region is highlighted in the viewport, and toggle on **ENCASTRE** in the **Edit Boundary Condition** dialog box that appears. Click **OK** to apply the boundary condition.

   Arrows appear on the face indicating the constrained degrees of freedom. The encastre boundary condition constrains all active structural degrees of freedom in the region specified; after the part is meshed and the job is created, this constraint will be applied to all the nodes that occupy the region.

Partition a dependent part instance

1. In the Model Tree, double-click the **Lug** item in the **Parts** container to make it current.

2. Use the **Partition Cell: Define Cutting Plane** tool  to divide the part in half. Use the **3 Points** method to define the cutting plane. When you are prompted to select a point, Abaqus/CAE highlights the points you can select: vertices, datum points, edge midpoints, or arc centers. In this model the points used to define the cutting plane are indicated in *Figure 2*. Again, you may need to rotate the view to facilitate your selection.

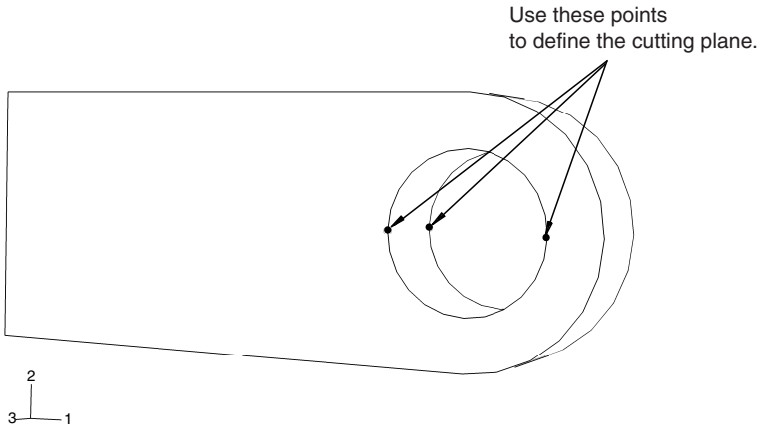


Figure 2: Points used to define the cutting plane.

3. Click **Create Partition** in the prompt area after you have finished selecting the points.

Apply a pressure load

1. In the Model Tree, double-click the **Loads** container to prescribe the pressure load. In the **Create Load** dialog box that appears, name the load `Pressure load` and select `LugLoad` as the step in which it will be applied. Select **Mechanical** as the category and **Pressure** as the type. Click **Continue**.
2. Select the surface associated with the bottom half of the hole using the cursor; the region is highlighted in *Figure 3*. When the appropriate surface is selected, click **Done** in the prompt area.

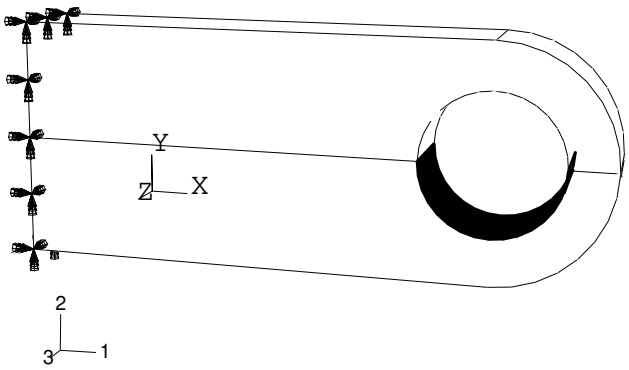


Figure 3: Surface to which pressure will be applied.

3. Specify a uniform pressure of `5.0E7` in the **Edit Load** dialog box, accept the default **Amplitude**, and click **OK** to apply the load.

   Arrows appear on the bottom half of the hole indicating the applied load.

Designing the mesh: partitioning and creating the mesh

You need to consider the type of element that will be used before you start building the mesh for a particular problem. For example, a suitable mesh design that uses quadratic elements may very well be unsuitable if you change to linear, reduced-integration elements.

For this example use 20-node hexahedral elements with reduced integration (C3D20R). Once you have selected the element type, you can design the mesh for the connecting lug. The most important decision regarding the mesh design for this example is how many elements to use around the circumference of the lug's hole. A possible mesh for the connecting lug is shown in *Figure 1*.

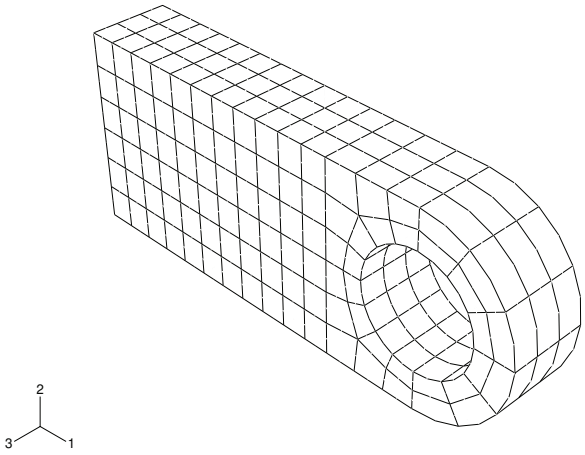


Figure 1: Suggested mesh of C3D20R elements for the connecting lug model.

Another thing to consider when designing a mesh is the type of results you want from the simulation. The mesh in *Figure 1* is rather coarse and, therefore, unlikely to yield accurate stresses. Four quadratic elements per 90° is the minimum number that should be considered for a problem such as this one; using twice that many is recommended to obtain reasonably accurate stress results. However, this mesh should be adequate to predict the overall level of deformation in the lug under the applied loads, which is what you were asked to determine. The influence of increasing the mesh density used in this simulation is discussed in *Mesh convergence*.

Abaqus/CAE offers a variety of meshing techniques to mesh models of different topologies. The different meshing techniques provide varying levels of automation and user control. The following three types of mesh generation techniques are available:

Structured meshing

Structured meshing applies preestablished mesh patterns to particular model topologies. Complex models must generally be partitioned into simpler regions to use this technique.

Swept meshing

Swept meshing extrudes an internally generated mesh along a sweep path or revolves it around an axis of revolution. Like structured meshing, swept meshing is limited to models with specific topologies and geometries.

Free meshing

The free meshing technique is the most flexible meshing technique. It uses no preestablished mesh patterns and can be applied to almost any model shape.

Bottom-up meshing

You use the bottom-up meshing technique to create a hexahedral or hex-dominated mesh on a solid region that is unmeshable or difficult to mesh using the automated top-down meshing techniques. Bottom-up meshing is a manual process that allows you to select the method and parameters that Abaqus/CAE uses to build up a solid mesh of hexahedral elements. Bottom-up meshing is not discussed in any of the examples in this guide.

When you enter the Mesh module, Abaqus/CAE color codes regions of the model according to the methods it will use to generate a mesh:

- Green indicates that a region can be meshed using structured methods.
- Yellow indicates that a region can be meshed using sweep methods.
- Pink indicates that a region can be meshed using the free method.
- Tan indicates that the region can be meshed using the bottom-up method.
- Orange indicates that a region cannot be meshed using the default element shape assignment and must be partitioned further.

Dependent part instances are colored blue at the assembly level. You must switch to a part-level view to mesh a dependent part instance.

In this problem you will create a structured mesh. You will find that the model must first be partitioned further to use this mesh technique. After the partitions have been created, a global part seed will be assigned and the mesh will be created.

Partition the lug

1. In the Model Tree, expand the **Lug** item underneath the **Parts** container and double-click **Mesh** in the menu that appears.

   The part is colored yellow initially, indicating that with the default set of mesh controls, a hexahedral mesh can be created only using a swept mesh technique. Additional cell partitions are required to permit structured meshing. Two partitions will be created. The first partition permits structured meshing to be used, and the second improves the overall quality of the mesh.

   Note:

   The **Object** field that appears in the context bar automatically displays the part so that you can partition the geometry directly within the Mesh module. The ability to switch between individual parts and the model assembly within the same module is available only in the Mesh module. This feature allows you to partition both dependent and independent part instances in the same module for the purpose of meshing. In all other modules, partitioning must be done strictly at the part level for dependent instances (as was done earlier when the pressure load was applied) or at the assembly level for independent part instances.

2. Partition both regions of the lug vertically by defining a cutting plane through the three points shown in *Figure 2* (use **[Shift] + Click** to select both regions simultaneously).

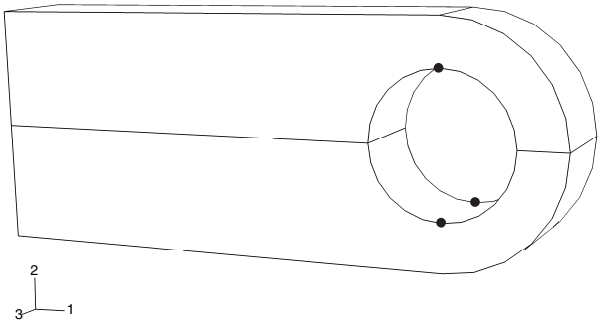


Figure 2: First partition to permit structured meshing.

3. Select **Tools->Datum** from the main menu bar, and create a datum point 0.075 m from the left end of the lug (as shown in *Figure 3*) using the **Offset from point** method.

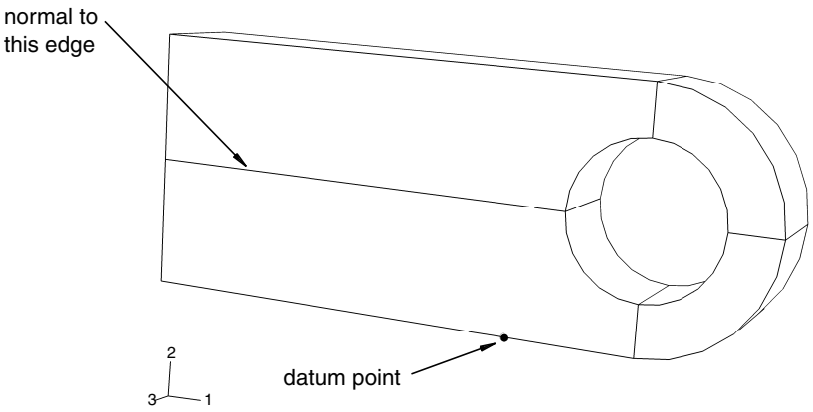


Figure 3: Second partition to improve the mesh quality.

4. Create the second vertical partition by using the **Point & Normal** method to define a cutting plane that is through the datum point you just created and normal to the centerline of the lug (as shown in *Figure 3*).

   The partitioned lug appears as in *Figure 4*. After you have partitioned the lug, all part regions should be colored green, which (based on the current mesh controls) indicates structured hexahedral element meshing will be used everywhere.

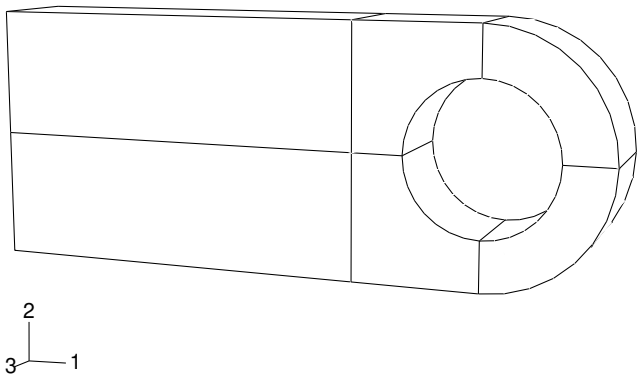


Figure 4: Final partitioned lug.

Assign a global part seed and create the mesh

1. From the main menu bar, select **Seed->Part**, and specify a target global element size of `0.007`. Seeds appear on all the edges.
2. From the main menu bar, select **Mesh->Element Type** to choose the element type for the part. Because of the partitions you have created, the part is now composed of several regions.
   a. Use the cursor to draw a box around the entire part and, thus, select all regions of the part. Click **Done** in the prompt area.
   b. In the **Element Type** dialog box that appears, select the **Standard** element library, **3D Stress** family, **Quadratic** geometric order, and **Hex**, **Reduced integration** element. Click **OK** to accept the choice of C3D20R as the element type.

   Note:

   If you are using the Abaqus Learning Edition, using second-order elements with a global seed size of 0.007 results in a mesh that exceeds the model size limits of the product. Either use first-order elements (C3D8R) with a global seed size of 0.007 or second-order elements with a global seed size of 0.01.

3. From the main menu bar, select **Mesh->Part**. Click **Yes** in the prompt area to mesh the part instance.

Creating, running, and monitoring a job

At this point the only task remaining to complete the model is defining the job. The job can then be submitted from within Abaqus/CAE and the solution progress monitored interactively.

Before continuing, rename the model to `Elastic` by clicking mouse button 3 on **Model-1** in the Model Tree and selecting **Rename** from the menu that appears. This model will later form the basis of the model used in the lug example discussed in *Materials*.

Save your model in a model database file named `Lug.cae`.

Create a job

1. In the Model Tree, double-click the **Jobs** container to create a job. Name the job `Lug`, and click **Continue**.
2. In the **Edit Job** dialog box, enter the following description: `Linear Elastic Steel Connecting Lug`.
3. Accept the default job settings, and click **OK**.

Run the job

1. In the Model Tree, click mouse button 3 on the job named **Lug** and select **Submit** to submit your job for analysis.

   A dialog box appears to warn you that history output has not been requested for the **LugLoad** step. Click **Yes** to continue with the job submission.
2. In the Model Tree, click mouse button 3 on the job named **Lug** and select **Monitor** from the menu that appears to open the job monitor.

   At the top of the dialog box, a summary of the solution progress is included. This summary is updated continuously as the analysis progresses. Any errors and/or warnings that are encountered during the analysis are noted in the appropriate tabbed pages. If any errors are encountered, correct the model and rerun the simulation. Be sure to investigate the cause of any warning messages and take appropriate action; recall that some warning messages can be ignored safely while others require corrective action.
3. When the job has completed, click **Dismiss** to close the job monitor.

Postprocessing—visualizing the results

In the Model Tree, click mouse button 3 on the job named **Lug** and select **Results** to enter the Visualization module and automatically open the output database (`.odb`) file created by this job. Alternatively, from the **Module** list located in the context bar, select **Visualization** to enter the Visualization module; open the `.odb` file by selecting **File->Open** from the main menu bar and double-clicking on the appropriate file.

In this section:

- *Plotting the deformed shape*
- *Changing the view*
- *Visible edges*
- *Render style*
- *Contour plots*
- *Displaying contour results on interior surfaces*
- *Maximum and minimum values*
- *Displaying a subset of the model*
- *Displaying a free body cut*
- *Generating tabular data reports for subsets of the model*

Plotting the deformed shape

You can plot the deformed shape.

From the main menu bar, select**Plot->Deformed Shape**; or use the  tool in the toolbox. *Figure 1* displays the deformed model shape at the end of the analysis. What is the displacement magnification level?

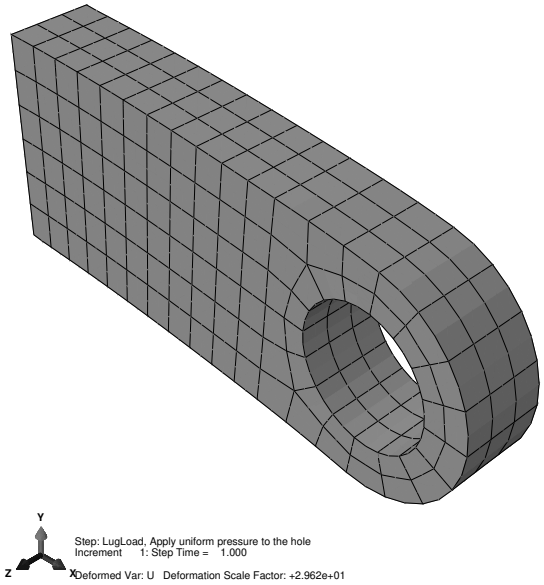


Figure 1: Deformed model shape of connecting lug (shaded).

Changing the view

The default view is isometric. You can change the view using the options in the **View** menu or the view tools in the View Manipulation toolbar. You can also specify a view by entering values for rotation angles, viewpoint, zoom factor, or fraction of viewport to pan. Direct view manipulation is also available using the 3D compass.

To manipulate the view using the 3D compass:

- Click and drag one of the straight axes of the 3D compass to pan along an axis.
- Click and drag any of the quarter-circular faces on the 3D compass to pan along a plane.
- Click and drag one of the three arcs along the perimeter of the 3D compass to rotate the model about the axis that is perpendicular to the plane containing the arc.
- Click and drag the free rotation handle (the point at the top of the 3D compass) to rotate the model freely about its pivot point.
- Click the label for any of the axes on the 3D compass to select a predefined view (the selected axis is perpendicular to the plane of the viewport).
- Double-click anywhere on the 3D compass to specify a view.

Most of the views in this guide are specified directly. This is to allow you to confirm the state of your model by checking against the images in the guide. You are encouraged, however, to practice using the above methods to manipulate your views as you deem fit.

1. From the main menu bar, select **View->Specify** (or double-click the 3D compass).

   The **Specify View** dialog box appears.

2. From the list of available methods, select **Viewpoint**.

   In the **Viewpoint** method, you enter three values representing the *X*-, *Y*-, and *Z*-position of an observer. You can also specify an up vector. Abaqus positions your model so that this vector points upward.

3. Enter the *X*-, *Y*-, and *Z*-coordinates of the viewpoint vector as `1, 1, 3` and the coordinates of the up vector as `0, 1, 0`.
4. Click **OK**.

   Abaqus/CAE displays your model in the specified view, as shown in *Figure 1*.

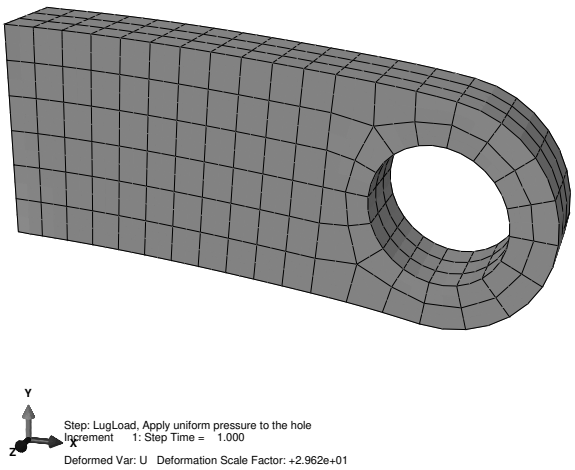


Figure 1: Deformed model shape viewed from specified viewpoint.

Visible edges

Several options are available for choosing which edges will be visible in the model display.

The previous plots show all exterior edges of the model; *Figure 1* displays only feature edges.

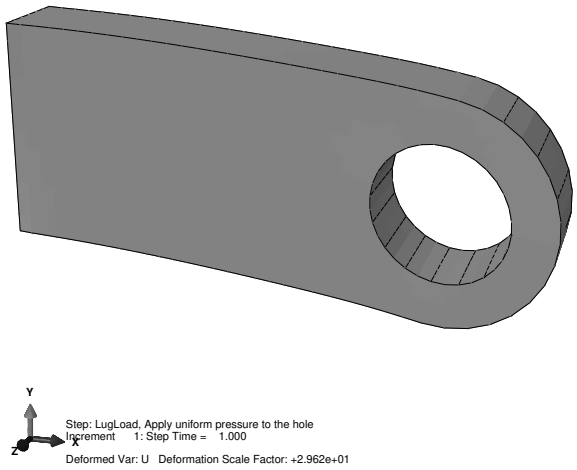


Figure 1: Deformed shape with only feature edges visible.

1. From the main menu bar, select **Options->Common**.
   The **Common Plot Options** dialog box appears.
2. Click the **Basic** tab if it is not already selected.
3. From the **Visible Edges** options, choose **Feature edges**.
4. Click **Apply**.
   The deformed plot in the current viewport changes to display only feature edges, as shown in *Figure 1*.

Render style

A shaded plot is a filled plot in which a lightsource appears to be directed at the model. This is the default render style and can be very useful when viewing complex three-dimensional models.

Three other render styles provide additional display options: wireframe, hidden line, and filled. You can select a render style from the **Common Plot Options** dialog box or from the tools on the Render Style toolbar: wireframe , hidden line , filled , and shaded . To display the wireframe plot shown in *Figure 1*, select **Exterior edges** in the **Common Plot Options** dialog box, click **OK** to close the dialog box, and select wireframe plotting by clicking the  tool. All subsequent plots will be displayed in the wireframe render style until you select another render style.

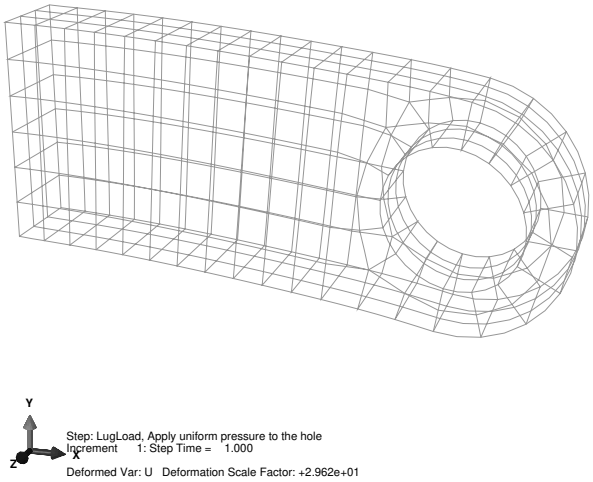


Figure 1: Wireframe plot.

A wireframe model showing internal edges can be visually confusing for complex three-dimensional models. You can use the other render style tools to select the hidden line and filled render styles, shown in *Figure 2* and *Figure 3*, respectively. These render styles are more useful when viewing complex three-dimensional models.

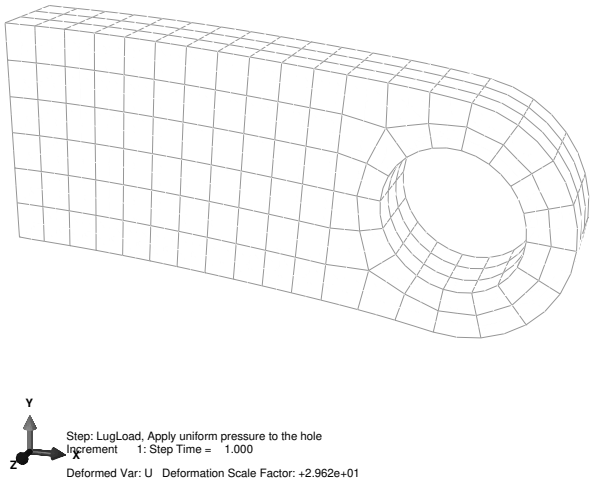


Figure 2: Hidden line plot.

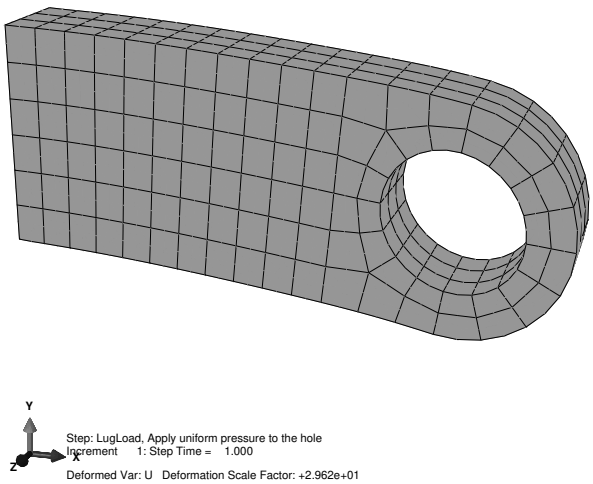


Figure 3: Filled element plot.

Contour plots

Contour plots display the variation of a variable across the surface of a model. You can create filled or shaded contour plots of field output results from the output database.

Generate a contour plot of the Mises stress

1. From the main menu bar, select **Plot->Contours->On Deformed Shape**.

   The filled contour plot shown in *Figure 1* appears.

   The Mises stress, `S Mises`, indicated in the legend title is the default variable chosen by Abaqus for this analysis. You can select a different variable to plot.

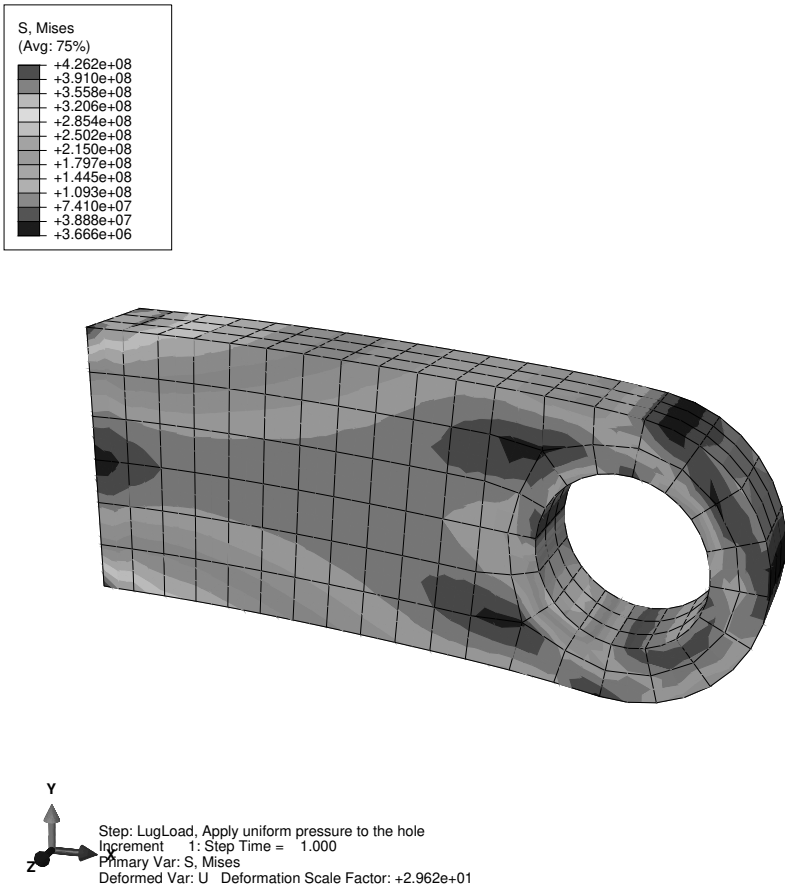


Figure 1: Filled contour plot of Mises stress.

2. From the main menu bar, select **Result->Field Output**.

   The **Field Output** dialog box appears; by default, the **Primary Variable** tab is selected.

3. From the list of available output variables, select a new variable to plot.

   Tip: You can also use the **Field Output** toolbar, located above the viewport, to change the displayed field output variable. For more information, see *Using the field output toolbar*.

4. Click **OK**.

The contour plot in the current viewport changes to reflect your selection.

Generate a customized contour plot

Abaqus/CAE offers many options to customize contour plots. To see the available options, click the **Contour Options**  tool in the toolbox. By default, Abaqus/CAE automatically computes the minimum and maximum values shown in your contour plots and evenly divides the range between these values into 12 intervals. You can control the minimum and maximum values Abaqus/CAE displays (for example, to examine variations within a fixed set of bounds), as well as the number of intervals.

1. In the **Basic** tabbed page of the **Contour Plot Options** dialog box, drag the **Contour Intervals** slider to change the number of intervals to nine.
2. In the **Limits** tabbed page of the **Contour Plot Options** dialog box, choose **Specify** beside **Max**; then enter a maximum value of `400E+6`.
3. Choose **Specify** beside **Min**; then enter a minimum value of `60E+6`.
4. Click **OK**.

   Abaqus/CAE displays your model with the specified contour option settings, as shown in *Figure 2* (this figure shows Mises stress; your plot will show whichever output variable you have chosen). These settings remain in effect for all subsequent contour plots until you change them or reset them to their default values.

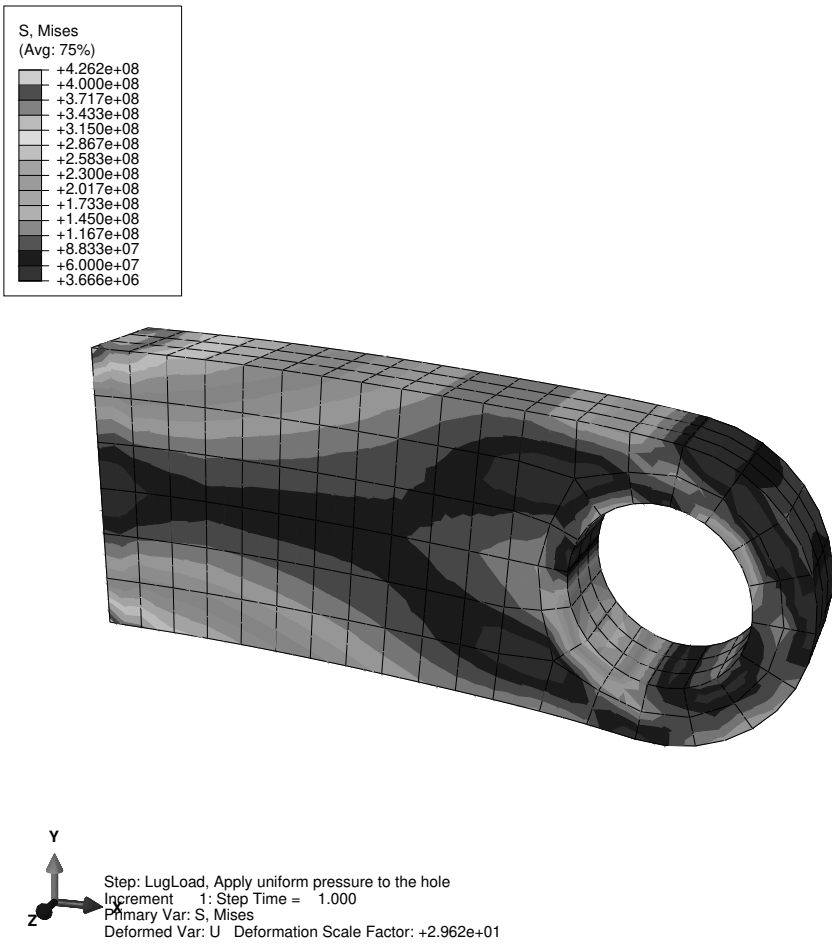


Figure 2: Customized plot of Mises stress.

Displaying contour results on interior surfaces

You can cut your model such that interior surfaces are made visible. For example, you may want to examine the stress distribution in the interior of a part. View cuts can be created for such purposes. Here, a simple planar cut is made through the lug to view the Mises stress distribution through the thickness of the part.

Note:

View cuts are also available throughout the other modules of Abaqus/CAE. They are illustrated here for convenience.

1. From the main menu bar, select **Tools->View Cut->Create**.
2. In the dialog box that appears, accept the default name and shape. Enter `0,0,0` as the **Origin** of the plane (i.e., a point through which the plane will pass), `1,0,1` as the **Normal axis** to the plane, and `0,1,0` as **Axis 2** of the plane.
3. Click **OK** to close the dialog box and to make the view cut.

   The view appears as shown in *Figure 1*.

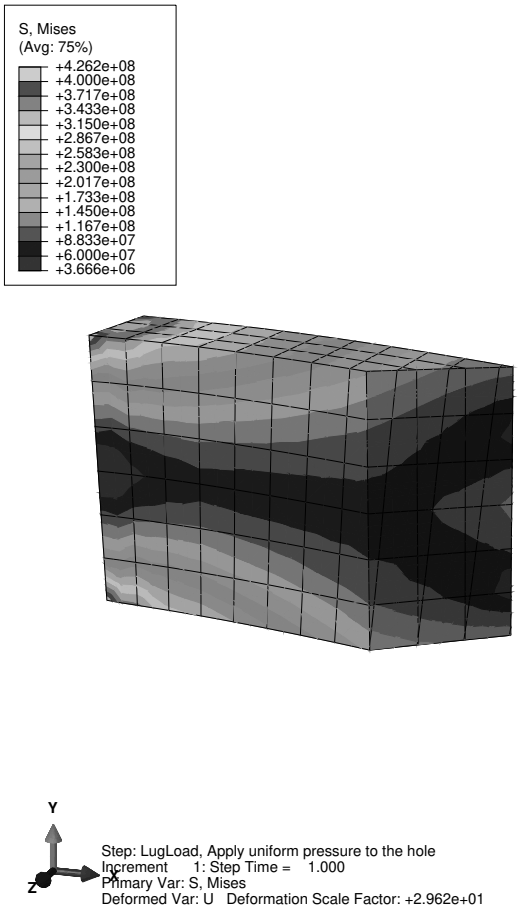


Figure 1: Mises stress through the lug thickness.

From the main menu bar, select **Tools->View Cut->Manager** to open the **View Cut Manager**. By default, the regions on and below the cut are displayed (as indicated by the check marks beneath the

on cut  and **below cut**  symbols). To translate or rotate the cut, choose **Translate** or **Rotate** from the list of available motions and enter a value or drag the slider at the bottom of the **View Cut Manager**.

4. To view the full model again, toggle off **Cut-4** in the **View Cut Manager**.

   For more information on view cuts, see *Cutting through a model*.

Maximum and minimum values

The maximum and minimum values of a variable in a model can be determined easily.

Display the minimum and maximum values of a contour variable

1. From the main menu bar, select **Viewport->Viewport Annotation Options**; then click the **Legend** tab in the dialog box that appears.

   The **Legend** options become available.
2. Toggle on **Show min/max values**.
3. Click **OK**.

   The contour legend changes to report the minimum and maximum contour values.

Contour the displacement of the connecting lug in the 2-direction

One of the goals of this example is to determine the deflection of the lug in the negative 2-direction. You can contour the displacement component of the lug in the 2-direction to determine its peak displacement in the vertical direction as follows. In the **Contour Plot Options** dialog box, click **Defaults** to reset the minimum and maximum contour values and the number of intervals to their default values before proceeding.

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Primary** if it is not already selected.

   Tip: You can click  on the left side of the **Field Output** toolbar to make your selections from the **Field Output** dialog box instead of the toolbar. If you use the dialog box, you must click **Apply** or **OK** for Abaqus/CAE to display your selections in the viewport.
2. From the list of available output variables in the center of the toolbar, select output variable **U**.
3. From the list of available components and invariants on the right side of the **Field Output** toolbar, select **U2**.

What is the maximum displacement value in the negative 2-direction?

Displaying a subset of the model

By default, Abaqus/CAE displays your entire model; however, you can choose to display a subset of your model called a display group. This subset can contain any combination of part instances, geometry (cells, faces, or edges), elements, nodes, and surfaces from the current model or output database. For the connecting lug model you will create a display group consisting of the elements at the bottom of the hole. Since a pressure load was applied to this region, an internal set is created by Abaqus that can be used for visualization purposes.

Display a subset of the model

1. In the Results Tree, double-click **Display Groups**.

   The **Create Display Group** dialog box opens.
2. From the **Item** list, select **Elements**. From the **Method** list, select **Internal sets**.

   Once you have selected these items, the list on the right-hand side of the **Create Display Group** dialog box shows the available selections.
3. Using this list, identify the set that contains the elements at the bottom of the hole. Toggle on **Highlight items in viewport** below the list so that the outlines of the elements in the selected set are highlighted in red.
4. When the highlighted set corresponds to the group of elements at the bottom of the hole, click **Replace**  to replace the current model display with this element set.

   Abaqus/CAE displays the specified subset of your model.
5. Click **Dismiss** to close the **Create Display Group** dialog box.

Display the face identification labels and element numbers on the undeformed model shape

When creating an Abaqus model, you may want to determine the face labels for a solid element. For example, you may want to verify that the correct load ID was used when applying pressure loads or when defining surfaces for contact. In such situations you can use the Visualization module to display the mesh after you have run a **datacheck** analysis that creates an output database file.

1. From the main menu bar, select **Options->Common**.

   The **Common Plot Options** dialog box appears.
2. Set the render style to filled; all visible element edges will be displayed for convenience.
   a. Toggle on **Filled** under **Render Style**.
   b. Toggle on **All edges** under **Visible Edges**.
3. Click the **Labels** tab, and toggle on **Show element labels** and **Show face labels**.
4. Click **Apply** to apply the plot options.
5. From the main menu bar, select **Plot->Undeformed Shape**; or use the  tool in the toolbox.

   Abaqus/CAE displays the element and face identification labels in the current display group.
6. Click **Defaults** in the **Common Plot Options** dialog box to restore the default plot settings and then click **OK** to close the dialog box.

Displaying a free body cut

You can define a free body cut to view the resultant forces and moments transmitted across a selected surface of a model. Force vectors are displayed with a single arrowhead and moment vectors with a double arrowhead.

1. To display the entire model in the viewport, click  in the **Display Group** toolbar.
2. From the main menu bar, select **Tools->Free Body Cut->Manager**.
3. Click **Create** in the **Free Body Cut Manager**.
4. From the dialog box that appears, select **3D element faces** as the **Selection method** and click **Continue**.
5. In the **Free Body Cross-Section** dialog box, select **Surfaces** as the **Item** and **Pick from viewport** as the **Method**.
6. In the prompt area, set the selection method to **by angle** and accept the default angle.
7. Select the surface, highlighted in *Figure 1* to define the free body cut cross-section.

   a. From the **Selection** toolbar, toggle off the **Select the Entity Closest to the Screen** tool  and ensure that the **Select From All Entities** tool  is selected.

   b. As you move the cursor in the viewport, Abaqus/CAE highlights all of the potential selections and adds ellipsis marks (...) next to the cursor arrow to indicate an ambiguous selection. Position the cursor so that one of the faces of the desired surface is highlighted, and click to display the first surface selection.

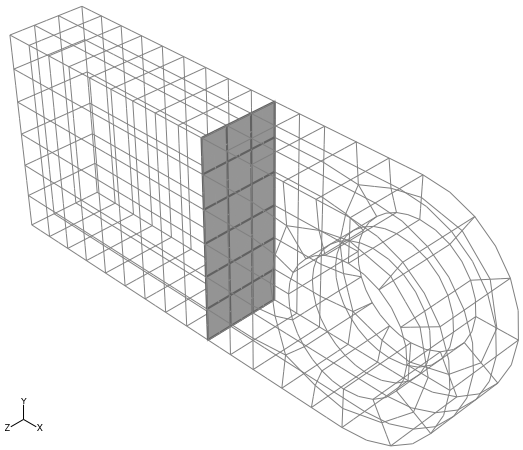


Figure 1: The free body cross-section.

   c. Use the **Next** and **Previous** buttons to cycle through the possible selections until the appropriate vertical surface is highlighted, and click **OK**.

8. Click **Done** in the prompt area to indicate your selection is complete. Click **OK** in the **Free Body Cross-Section** dialog box.
9. In the **Edit Free Body Cut** dialog box, accept the default settings for the **Summation Point** and the **Component Resolution**. Click **OK** to close the dialog box.
10. Click **Options** in the **Free Body Cut Manager**.
11. From the **Free Body Plot Options** dialog box, select the **Force** tab in the **Color & Style** tabbed page. Click the resultant color sample  to change the color of the resultant force arrow.
12. Once you have selected a new color for the resultant force arrow, click **OK** in the **Free Body Plot Options** dialog box and click **Dismiss** in the **Free Body Cut Manager**.

The free body cut is displayed in the viewport, as shown in *Figure 2*.

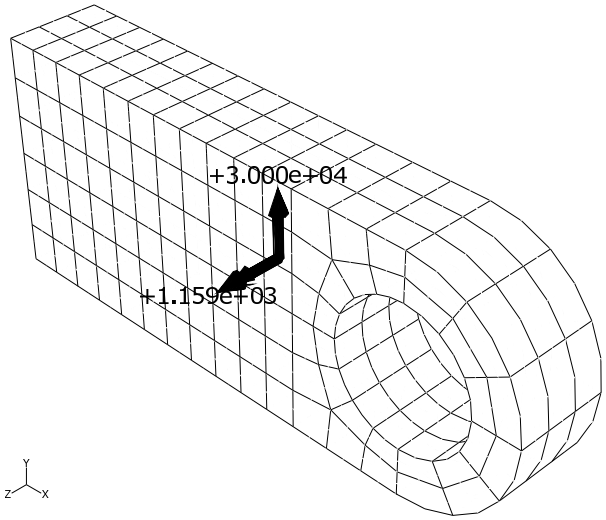


Figure 2: Free body cut displayed on the connecting lug.

Generating tabular data reports for subsets of the model

Tabular output data can be generated for selected regions of the model very easily using Abaqus/CAE. This is achieved using display groups in conjunction with the report generation feature. For the connecting lug problem we will generate the following tabular data reports:

- Stresses in the elements at the built-in end of the lug (to determine the maximum stress in the lug)
- Reaction forces at the built-in end of the lug (to check that the reaction forces at the constraints balance the applied loads)
- Vertical displacements at the bottom of the hole (to determine the deflection of the lug when the load is applied)

Since we did not create geometry sets corresponding to these regions, each of these reports will be generated using display groups whose contents are selected in the viewport. Thus, begin by creating and saving display groups for each region of interest.

Then generate the reports.

Create and save a display group containing the elements at the built-in end

1. In the Results Tree, double-click **Display Groups**.
2. Choose **Elements** from the **Item** list and **Pick from viewport** as the selection method.
3. Restore the option to select entities closest to the screen.
4. In the prompt area, set the selection method to **by angle**; and click the built-in face of the lug. Click **Done** when all the elements at the built-in face of the lug are highlighted in the viewport. In the **Create Display Group** dialog box, click **Save Selection As**. Save the display group as `built-in elements`.

Create and save a display group containing the nodes at the built-in end

1. In the **Create Display Group** dialog box, choose **Nodes** from the **Item** list and **Pick from viewport** as the selection method.
2. In the prompt area, set the selection method to **by angle**; and click the built-in face of the lug. Click **Done** when all the nodes on the built-in face of the lug are highlighted in the viewport. In the **Create Display Group** dialog box, click **Save Selection As**. Save the display group as `built-in nodes`.

Create and save a display group containing the nodes at the bottom of the hole

1. In the **Create Display Group** dialog box, click **Edit Selection** to select a different group of nodes.
2. In the prompt area, set the selection method to **individually**; and select the nodes at the bottom of the hole in the lug, as indicated in *Figure 1*. Click **Done** when all the nodes on the bottom of the hole are highlighted in the viewport. In the **Create Display Group** dialog box, click **Save Selection As**. Save the display group as `nodes at hole bottom`.

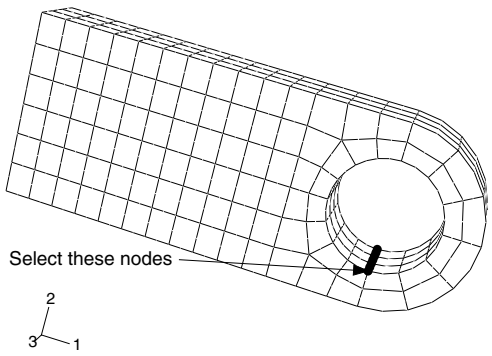


Figure 1: Nodes in display group `nodes at hole bottom`.

Generate field data reports

1. In the Results Tree, click mouse button 3 on **built-in elements** underneath the **Display Groups** container. In the menu that appears, select **Plot** to make it the current display group.
2. From the main menu bar, select **Report->Field Output**.
3. In the **Variable** tabbed page of the **Report Field Output** dialog box, accept the default position labeled **Integration Point**. Click the triangle next to **S: Stress components** to expand the list of available variables. From this list, select **Mises** and the six individual stress components: **S11**, **S22**, **S33**, **S12**, **S13**, and **S23**.
4. In the **Setup** tabbed page, name the report `Lug.rpt`. In the **Data** region at the bottom of the page, toggle off **Column totals**.
5. Click **Apply**.
6. In the Results Tree, click mouse button 3 on **built-in nodes** underneath the **Display Groups** container. In the menu that appears, select **Plot** to make it the current display group. (To see the nodes, toggle on **Show node symbols** in the **Common Plot Options** dialog box.)
7. In the **Variable** tabbed page of the **Report Field Output** dialog box, change the position to **Unique Nodal**. Toggle off **S: Stress components**, and select **RF1**, **RF2**, and **RF3** from the list of available **RF: Reaction force** variables.
8. In the **Data** region at the bottom of the **Setup** tabbed page, toggle on **Column totals**.
9. Click **Apply**.
10. In the Results Tree, click mouse button 3 on **nodes at hole bottom** underneath the **Display Groups** container. In the menu that appears, select **Plot** to make it the current display group.
11. In the **Variable** tabbed page of the **Report Field Output** dialog box, toggle off **RF: Reaction force**, and select **U2** from the list of available **U: Spatial displacement** variables.
12. In the **Data** region at the bottom of the **Setup** tabbed page, toggle off **Column totals**.
13. Click **OK**.

    Open the file `Lug.rpt` in a text editor. A portion of the table of element stresses is shown below. The element data are given at the element integration points. The integration point associated with a given element is noted under the column labeled `Int Pt`. The bottom of the table contains information on the maximum and minimum stress values in this group of elements. The results indicate that the maximum Mises stress at the built-in end is approximately 330 MPa. Your results may differ slightly if your mesh is not identical to the one used here.

```
Field Output Report

Source 1
```

```
---------

   ODB: Lug.odb
   Step: LugLoad
   Frame: Increment      1: Step Time =    1.000


Loc 1 : Integration point values from source 1

Output sorted by column "Element Label".

Field Output reported at integration points for part: LUG-1

Element   Int      S.Mises        S.S11        S.S22        S.S33
      S.S12
Label     Pt        @Loc 1       @Loc 1       @Loc 1       @Loc 1
     @Loc 1
------------------------------------------------------------------------------

        S.S13        S.S23
       @Loc 1       @Loc 1
--------------------------
   31     1    84.0567E+06  76.2075E+06   14.021E+06 -274.446E+03
-26.4339E+06
 -159.756E+03  1.70193E+06
   31     2    88.1108E+06  79.9769E+06  16.7815E+06  5.07167E+06
-30.8241E+06
    3.045E+06  2.20251E+06
   31     3    71.0378E+06  87.4511E+06  30.5433E+06  22.7759E+06
-11.8374E+06
   17.114E+06  1.47033E+06
   31     4    64.3978E+06  76.4473E+06  26.1412E+06  19.0617E+06
-18.6815E+06
  7.36221E+06  411.436E+03
   31     5    48.4688E+06  20.2162E+06  4.20669E+06  68.3536E+03
-25.8284E+06
 -78.8286E+03  1.65403E+06
   31     6    55.6407E+06  21.1693E+06  4.87661E+06  1.46182E+06
-30.2661E+06
  796.514E+03   2.0911E+06
   31     7    25.1732E+06  22.9107E+06  7.96147E+06  5.89815E+06
-10.1069E+06
  4.55755E+06  1.45446E+06
   31     8    32.8268E+06   19.928E+06  6.76018E+06  4.88587E+06
-16.9708E+06
   1.9631E+06  378.549E+03
    .
    .
  198     1    239.655E+06 -247.046E+06 -20.9897E+06 -13.4824E+06
-38.331E+06
  7.48079E+06  -1.1505E+06
  198     2    237.839E+06 -235.534E+06 -13.8995E+06  2.83029E+06
-33.891E+06
 -1.25472E+06 -1.57941E+06
  198     3    190.507E+06 -228.049E+06 -53.1236E+06 -51.1938E+06
-37.0096E+06
  20.3556E+06 -419.555E+03
  198     4    219.144E+06 -259.016E+06 -65.4028E+06  -61.292E+06
-30.1529E+06
   48.227E+06 -2.60239E+06
  198     5    322.607E+06 -330.074E+06 -2.82071E+06 -14.7802E+06
```

```
-12.9162E+06
   9.1461E+06 -189.416E+03
  198     6    320.169E+06 -316.345E+06  1.80108E+06  4.87007E+06
-9.15387E+06
 -4.10769E+06 -1.03378E+06
  198     7    300.073E+06 -364.931E+06 -95.1687E+06 -93.2852E+06
-69.7659E+06
  26.8241E+06 -343.386E+03
  198     8    331.098E+06 -399.004E+06 -109.695E+06 -104.075E+06
-62.7377E+06
  64.3938E+06 -2.63754E+06


  Minimum      25.1732E+06 -399.004E+06 -109.695E+06 -122.144E+06
-72.2982E+06
 -64.4051E+06 -2.63899E+06
      At Element        33          196          198          197
         98
           99           99
          Int Pt         8            7            8            8
          4
            4            4
  Maximum      331.159E+06  399.073E+06  109.719E+06  122.172E+06
-9.15387E+06
  64.4051E+06    2.639E+06
      At Element        97           99           99           98
        196
           97           97
          Int Pt         3            4            4            4
          5
            3            3
```

How does the maximum value of Mises stress compare to the value reported in the contour plot generated earlier? Do the two maximum values correspond to the same point in the model? The Mises stresses shown in the contour plot have been extrapolated to the nodes, whereas the stresses written to the report file for this problem correspond to the element integration points. Therefore, the location of the maximum Mises stress in the report file is not exactly the same as the location of the maximum Mises stress in the contour plot. This difference can be resolved by requesting that stress output at the nodes (extrapolated from the element integration points and averaged over all elements containing a given node) be written to the report file. If the difference is large enough to be of concern, this is an indication that the mesh may be too coarse.

The table listing the reaction forces at the constrained nodes is shown below. The `Total` entry at the bottom of the table contains the net reaction force components for this group of nodes. The results confirm that the total reaction force in the 2-direction at the constrained nodes is equal and opposite to the applied load of −30 kN in that direction.

```
Field Output Report

Source 1
---------

   ODB: Lug.odb
   Step: LugLoad
   Frame: Increment      1: Step Time =    1.000

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".
```

```
Field Output reported at nodes for part: LUG-1
            Node          RF.RF1          RF.RF2          RF.RF3
           Label          @Loc 1          @Loc 1          @Loc 1
-------------------------------------------------------------------
               3    -60.6106E-03        -118.486    -31.3131E-03
               4    -60.6102E-03        -118.486     31.3133E-03
               6         538.194         289.574         382.416
               7         538.194         289.574        -382.416
              11        -538.255         289.563        -382.329
       .
       .
            1334     5.90186E+03         216.099     1.63004E+03
            1336     6.60254E+03     1.81494E+03     -45.543E-06
            1337     9.81613E+03         692.328        -791.954
            1339     6.35335E+03      1.7276E+03        -331.533
            1340     5.90186E+03         216.098    -1.63004E+03


  Minimum           -9.81734E+03        -264.368    -1.63023E+03
         At Node             953             258             950

  Maximum            9.81613E+03     1.81556E+03     1.63022E+03
         At Node            1337             951             956

           Total    -912.953E-06     30.0000E+03    -46.7648E-06
```

The table showing the displacements of the nodes along the bottom of the hole (listed below) indicates that the bottom of the hole in the lug has displaced about 0.3 mm.

```
Field Output Report

Source 1
---------

   ODB: Lug.odb
   Step: LugLoad
   Frame: Increment      1: Step Time =    1.000

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".

Field Output reported at nodes for part: LUG-1

            Node            U.U2
           Label          @Loc 1
--------------------------------
              23    -314.158E-06
              24    -314.158E-06
             143      -314.2E-06
             144      -314.2E-06
            1522    -314.163E-06
            1526    -314.211E-06
            1529    -314.163E-06


  Minimum           -314.211E-06

         At Node            1526
  Maximum           -314.158E-06
```

```
         At Node              23
```

Rerunning the analysis using Abaqus/Explicit

You will now evaluate the dynamic response of the lug when the same load is applied suddenly. Of special interest is the transient response of the lug. You will have to modify the model for the Abaqus/Explicit analysis. Before proceeding, copy the existing model to a new model named `Explicit`. Make all subsequent changes to the `Explicit` model (you may want to collapse the **Elastic** model to avoid confusion). Before running the job you will need to add a density definition to the material model, change the step type, and change the element type. In addition, you should make modifications to the field output requests.

1. Edit the material definition for **Steel** to include a mass density of `7800`.
2. Replace the static step named **LugLoad** with a dynamic, explicit step. Enter `Dynamic lug loading` as the step description, and enter `0.005` s for the time period of the step.
3. Edit the field output request named **F-Output-1**. In the **Edit Field Output Request** dialog box, select **Evenly spaced time intervals** as the frequency type and enter `125` as the interval value.
4. Accept the default history output request.
5. Change the element type used to mesh the lug. In the **Element Type** dialog box, select the **Explicit** element library, **3D Stress** family, and **Linear** geometric order. In addition, select enhanced hourglass control for the **Hex** shape. The selected element type is C3D8R.
6. Create and submit a job named **expLug** using the model named `Explicit`.
7. Monitor the progress of the job.

   At the top of the **expLug Monitor** dialog box, a summary of the solution progress is included. This summary is updated continuously as the analysis progresses. Any errors and/or warnings that are encountered during the analysis are noted in the appropriate tabbed pages. If any errors are encountered, correct the model and rerun the simulation. Be sure to investigate the cause of any warning messages and take appropriate action; recall that some warning messages can be ignored safely while others require corrective action.

Postprocessing the dynamic analysis results

In the static analysis performed with Abaqus/Standard you examined the deformed shape of the lug as well as stress and displacement output. For the Abaqus/Explicit analysis you can similarly examine the deformed shape, stresses, and displacements in the lug. Because transient dynamic effects may result from a sudden loading, you should also examine the time histories for internal and kinetic energy, displacement, and Mises stress.

Open the output database (`.odb`) file created by this job.

Plotting the deformed shape

From the main menu bar, select**Plot->Deformed Shape**; or use the  tool in the toolbox. *Figure 1* displays the deformed model shape at the end of the analysis.

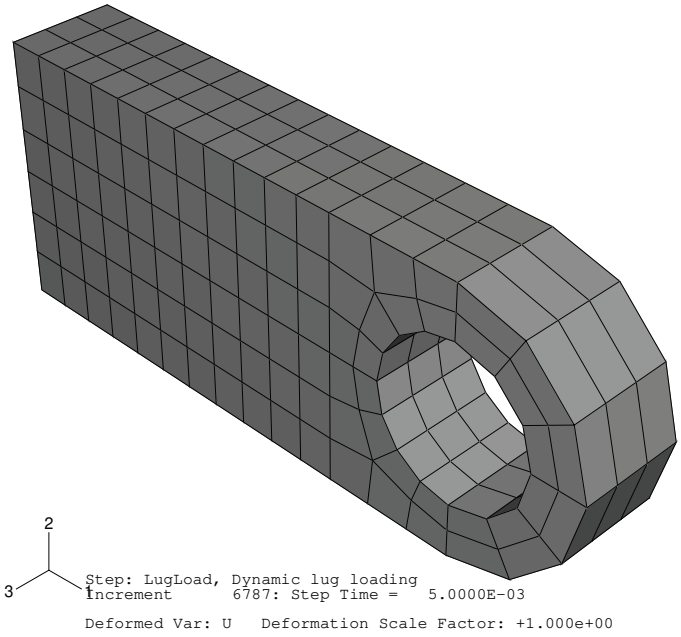


Figure 1: Deformed model shape for the explicit analysis (shaded).

As discussed earlier, Abaqus/Explicit assumes large deformation theory by default; thus, the deformation scale factor is automatically set to 1. If the displacements are too small to be seen, scaling can be applied to aid the study of the response.

To see the vibrations in the lug more clearly, change the deformation scale factor to 50. In addition, animate the time history of the deformed shape of the lug and decrease the frame rate of the time history animation.

The time history animation of the deformed shape of the lug shows that the suddenly applied load induces vibrations in the lug. Additional insights about the behavior of the lug under this type of loading can be gained by plotting the kinetic energy, internal energy, displacement, and stress in the lug as a function of time. Some of the questions to consider are:

1. Is energy conserved?
2. Was large-displacement theory necessary for this analysis?
3. Are the peak stresses reasonable? Will the material yield?

X–Y plotting

X–Y plots can display the variation of a variable as a function of time. You can create X–Y plots from field and history output.

To create X–Y plots of the internal and kinetic energy as a function of time:

1. In the Results Tree, expand the **History Output** container underneath the output database named `expLug.odb`.
2. The list of all the variables in the history portion of the output database appears; these are the only history output variables you can plot.

   From the list of available output variables, double-click **ALLIE** to plot the internal energy for the whole model.

   Abaqus reads the data for the curve from the output database file and plots the graph shown in *Figure 2*.

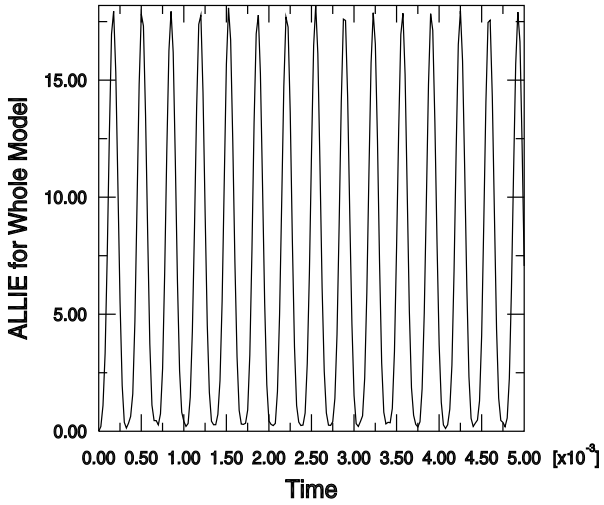


Figure 2: Internal energy for the whole model.

3. Repeat this procedure to plot **ALLKE**, the kinetic energy for the whole model (shown in *Figure 3*).

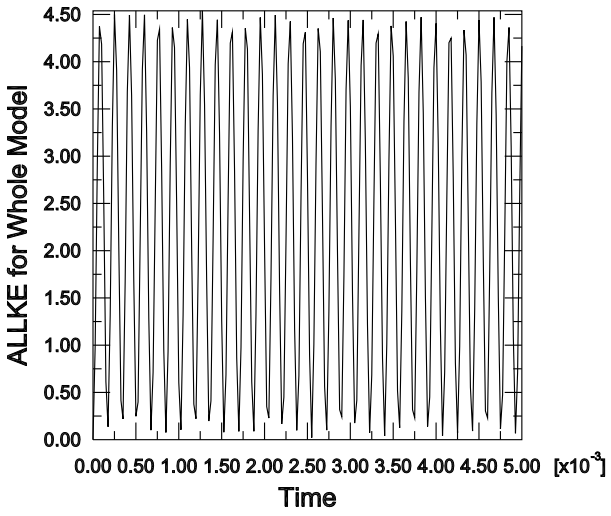


Figure 3: Kinetic energy for the whole model.

Both the internal energy and the kinetic energy show oscillations that reflect the vibrations of the lug. Throughout the simulation, kinetic energy is transformed into internal (strain) energy and vice-versa. Since the material is linear elastic, total energy is conserved. This can be seen by plotting `ETOTAL`, the total energy of the system, together with `ALLIE` and `ALLKE`. The value of `ETOTAL` is approximately zero throughout the course of the analysis. Energy balances in dynamic analysis are discussed further in *Nonlinear Explicit Dynamics*.

We will examine the nodal displacements at the bottom of the lug hole to evaluate the significance of geometrically nonlinear effects in this simulation.

To generate a plot of displacement versus time:

1. Plot the deformed shape of the lug. In the Results Tree, double-click **XY Data**.
2. In the **Create XY Data** dialog box that appears, select **ODB field output** as the source and click **Continue**.
3. In the **XY Data from ODB Field Output** dialog box that appears, select **Unique Nodal** as the type of position from which the *X*–*Y* data should be read.
4. Click the arrow next to **U: Spatial displacement** and toggle on **U2** as the displacement variable for the *X*–*Y* data.
5. Select the **Elements/Nodes** tab. Choose **Pick from viewport** as the selection method for identifying the node for which you want *X*–*Y* data.
6. Click **Edit Selection**. In the viewport, select one of the nodes on the bottom of the hole as shown in *Figure 4* (if necessary, change the render style to facilitate your selection). Click **Done** in the prompt area.

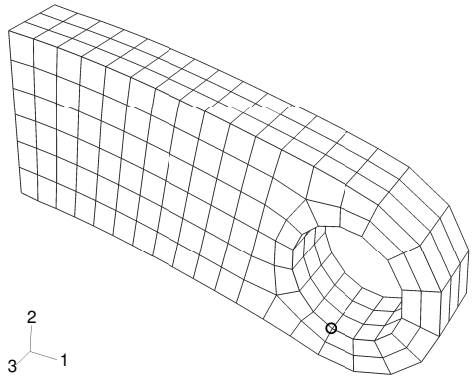


Figure 4: Selected node at the bottom of the hole.

7. Click **Plot** in the **XY Data from ODB Field Output** dialog box to plot the nodal displacement as a function of time.

   The history of the oscillation, as shown in *Figure 5*, indicates that the displacements are small (relative to the structure's dimensions).

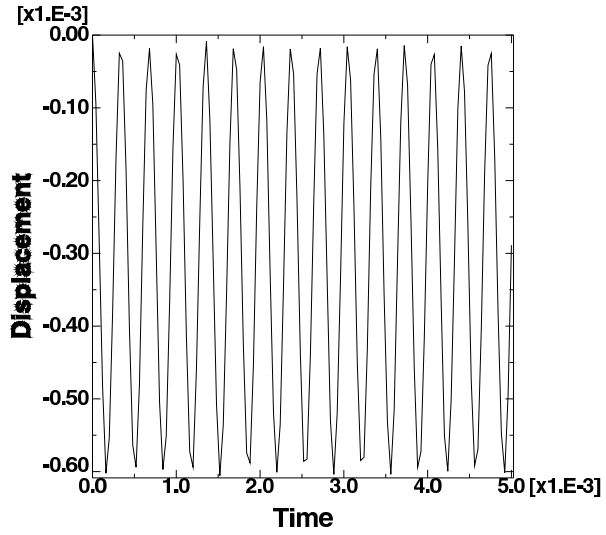


Figure 5: Displacement of a node at the bottom of the hole.

Thus, this problem could have been solved adequately using small-deformation theory. This would have reduced the computational cost of the simulation without significantly affecting the results. Nonlinear geometric effects are discussed further in *Nonlinearity*.

We are also interested in the stress history of the connecting lug. The area of the lug near the built-in end is of particular interest because the peak stresses expected to occur there may cause yielding in the material.

To generate a plot of Mises stress versus time:

1. Plot the deformed shape of the lug again.
2. Select the **Variables** tab in the **XY Data from ODB Field Output** dialog box. Deselect **U2** as the variable for the *X*–*Y* data plot.

3. Change the **Position** field to **Integration Point**.
4. Click the arrow next to **S: Stress components** and toggle on **Mises** as the stress variable for the *X*–*Y* data.
5. Select the **Elements/Nodes** tab. Choose **Pick from viewport** as the selection method for identifying the element for which you want X–Y data.
6. Click **Edit Selection**. In the viewport, select one of the elements near the built-in end of the lug as shown in *Figure 6*. Click **Done** in the prompt area.

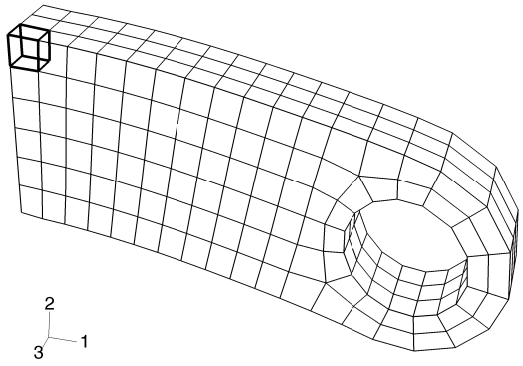


Figure 6: Selected element near the built-in end of the lug.

7. Click **Plot** in the **XY Data from ODB Field Output** dialog box to plot the Mises stress at the selected element as a function of time.

The peak Mises stress is on the order of 550 MPa, as shown in *Figure 7*. This value is larger than the typical yield strength of steel. Thus, the material would have yielded before experiencing such a large stress state. Material nonlinearity is discussed further in *Materials*.

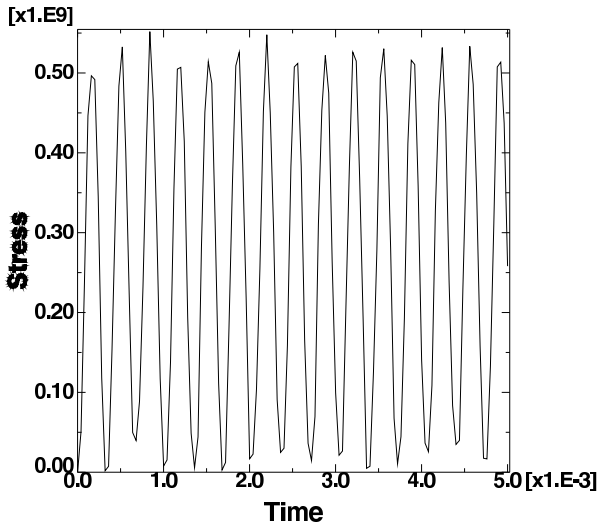


Figure 7: Mises stress near the built-in end of the lug.

Mesh convergence

It is important that you use a sufficiently refined mesh to ensure that the results from your Abaqus simulation are adequate. Coarse meshes can yield inaccurate results in analyses using implicit or explicit methods. The numerical solution provided by your model will tend toward a unique value as you increase the mesh density. The computer resources required to run your simulation also increase as the mesh is refined. The mesh is said to be converged when further mesh refinement produces a negligible change in the solution.

As you gain experience, you will learn to judge what level of refinement produces a suitable mesh to give acceptable results for most simulations. However, it is always good practice to perform a mesh convergence study, where you simulate the same problem with a finer mesh and compare the results. You can have confidence that your model is producing a mathematically accurate solution if the two meshes give essentially the same result.

Mesh convergence is an important consideration in both Abaqus/Standard and Abaqus/Explicit. The connecting lug will be used as an example of a mesh refinement study by further analyzing the connecting lug in Abaqus/Standard using four different mesh densities (*Figure 1*). The number of elements used in each mesh is indicated in the figure.

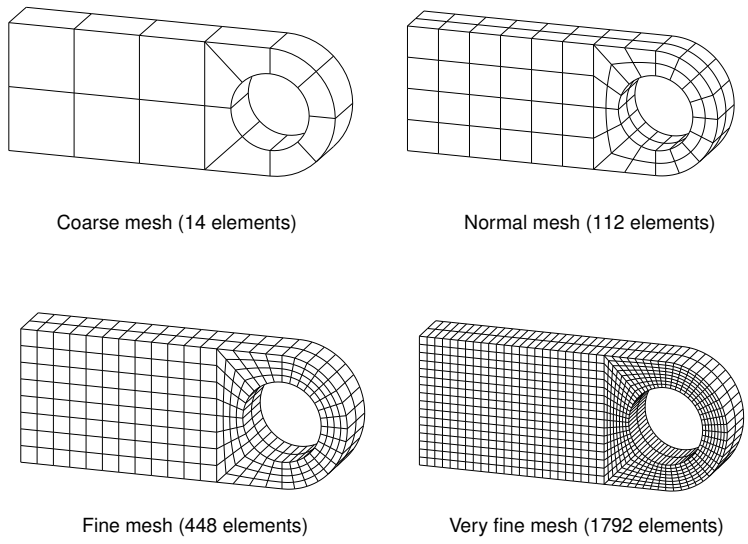


Figure 1: Different meshes for the connecting lug problem.

Note: The meshes in *Figure 1* can be obtained by further partitioning, as indicated in *Figure 2*, with appropriate mesh seeding.

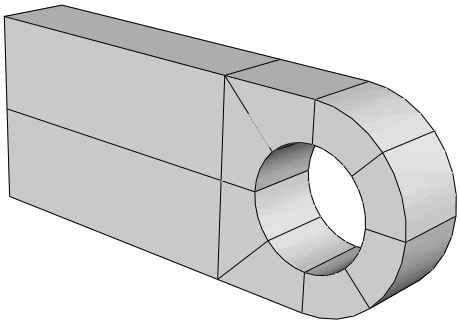


Figure 2: Additional partitions for the connecting lug.

We consider the influence of the mesh density on three particular results from this model:

- The displacement of the bottom of the hole.
- The peak Mises stress at the stress concentration on the bottom surface of the hole.
- The peak Mises stress where the lug is attached to the parent structure.

The locations where the results are compared are shown in *Figure 3*.

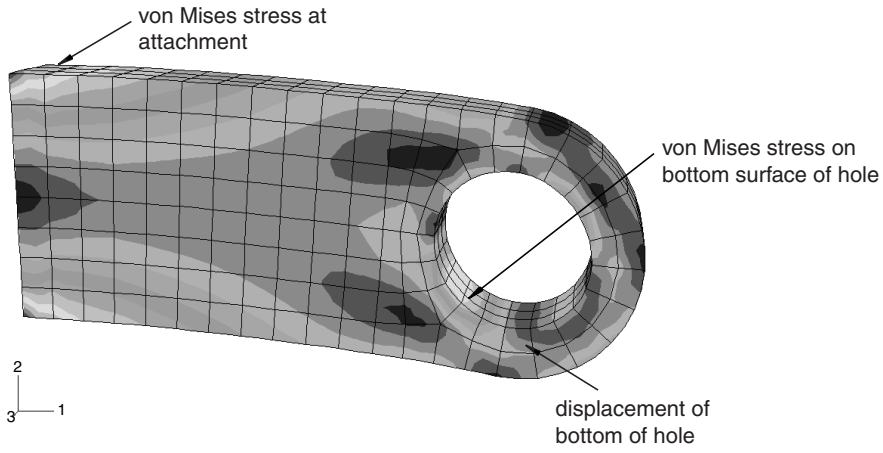


Figure 3: Locations where results are compared in the mesh refinement study.

The results for each of the four mesh densities are compared in *Table 1*, along with the CPU time required to run each simulation.

Table 1: Results of mesh refinement study.

Mesh	Displacement of bottom of hole	Stress at bottom of hole	Stress at attachment	Relative CPU time
Coarse	3.07E–4	256.E6	312.E6	0.83
Normal	3.13E–4	311.E6	365.E6	1.0
Fine	3.14E–4	332.E6	426.E6	3.2
Very fine	3.15E–4	345.E6	496.E6	13.3

The coarse mesh predicts less accurate displacements at the bottom of hole, but the normal, fine, and very fine meshes all predict similar results. The normal mesh is, therefore, converged as far as the displacements are concerned. The convergence of the results is plotted in *Figure 4*.

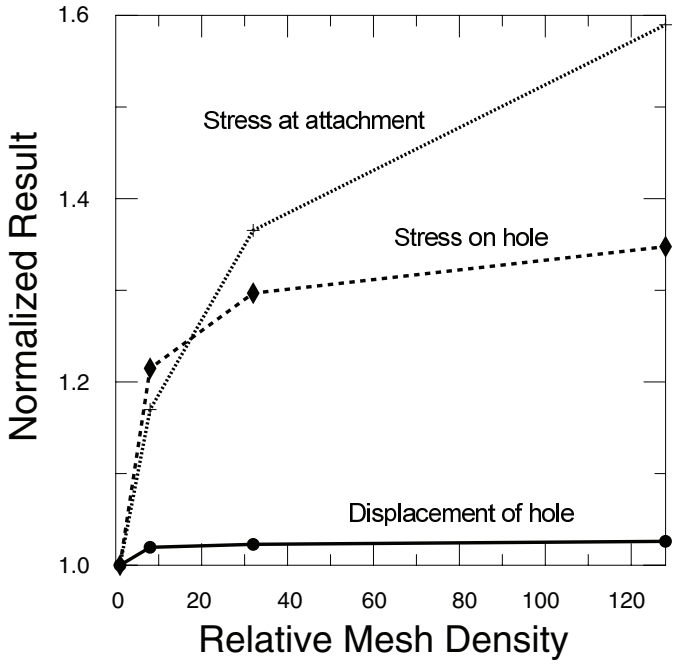


Figure 4: Convergence of results in mesh refinement study.

All the results are normalized with respect to the values predicted by the coarse mesh. The peak stress on the bottom of the hole converges much more slowly than the displacements because stress and strain are calculated from the displacement gradients; thus, a much finer mesh is required to predict accurate displacement gradients than is needed to calculate accurate displacements.

Mesh refinement significantly changes the stress calculated at the attachment of the connecting lug; it continues to increase with continued mesh refinement. A stress singularity exists at the corner of the lug where it attaches to the parent structure. Theoretically the stress is infinite at this location; therefore, increasing the mesh density will not produce a converged stress value at this location. This singularity occurs because of the idealizations used in the finite element model. The connection between the lug and the parent structure has been modeled as a sharp corner, and the parent structure has been modeled as rigid. These idealizations lead to the stress singularity. In reality there probably will be a small fillet between the lug and the parent structure, and the parent structure will be deformable, not rigid. If the exact stress in this location is required, the fillet between the components must be modeled accurately (see *Figure 5*) and the stiffness of the parent structure must also be considered.

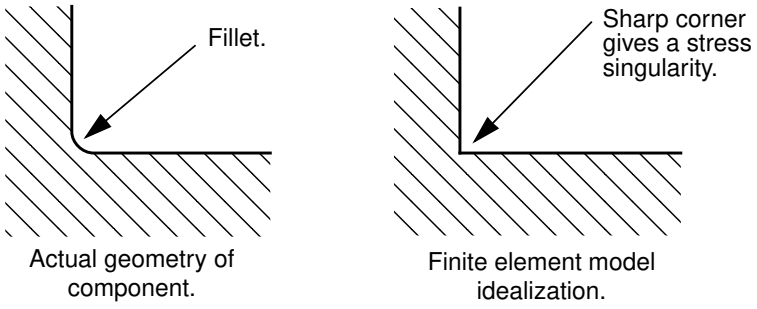


Figure 5: Idealizing a fillet as a sharp corner.

It is common to omit small details like fillet radii from a finite element model to simplify the analysis and to keep the model size reasonable. However, the introduction of any sharp corner into a model will lead to a stress singularity at that location. This normally has a negligible effect on the overall response of the model, but the predicted stresses close to the singularity will be inaccurate.

For complex, three-dimensional simulations the available computer resources often dictate a practical limit on the mesh density that you can use. In this case you must use the results from the analysis carefully. Coarse meshes are often adequate to predict trends and to compare how different concepts behave relative to each other. However, you should use the actual magnitudes of displacement and stress calculated with a coarse mesh with caution.

It is rarely necessary to use a uniformly fine mesh throughout the structure being analyzed. You should use a fine mesh mainly in the areas of high stress gradients and use a coarser mesh in areas of low stress gradients or where the magnitude of the stresses is not of interest. For example, *Figure 6* shows a mesh that is designed to give an accurate prediction of the stress concentration at the bottom of the hole.

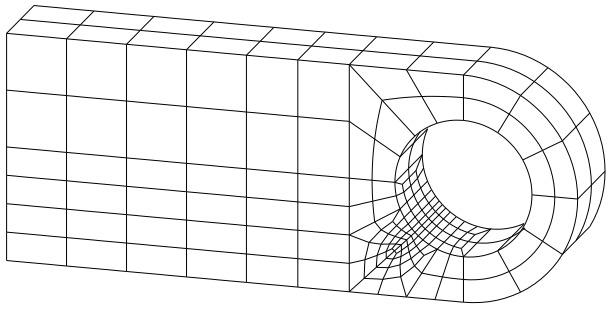


Figure 6: Mesh refined around the hole.

A fine mesh is used only in the region of high stress gradients, and a coarse mesh is used elsewhere. The results from an Abaqus/Standard simulation with this locally refined mesh are shown in *Table 2*. This table shows that the results are comparable to those from the very fine mesh, but the simulation with the locally refined mesh required considerably less CPU time than the analysis with the very fine mesh.

Table 2: Comparison of very fine and locally refined meshes.

Mesh	Displacement of bottom of hole	Stress at bottom of hole	Relative CPU time
Very fine	3.15E–4	345.E6	22.5
Locally refined	3.14E–4	346.E6	3.44

You can often predict the locations of the highly stressed regions of a model—and, hence, the regions where a fine mesh is required—using your knowledge of similar components or with hand calculations. This information can also be gained by using a coarse mesh initially to identify the regions of high stress and then refining the mesh in these regions. The latter procedure is carried out easily using preprocessors like Abaqus/CAE where the complete numerical model (i.e., material properties, boundary conditions, loads, etc.) can be defined based on the geometry of the structure. It is simple to mesh the geometry coarsely for the initial simulation and then to refine the mesh in the appropriate regions, as indicated by the results from the coarse simulation.

Abaqus provides an advanced feature, called submodeling, that allows you to obtain more detailed (and accurate) results in a region of interest in the structure. The solution from a coarse mesh of the entire structure is used to “drive” a detailed local analysis that uses a fine mesh in this region of interest. (This topic is beyond the scope of this guide. See *About Submodeling* for further details.)

Related Abaqus examples

If you are interested in learning more about using continuum elements in Abaqus, you should examine the following problems:

- *Geometrically nonlinear analysis of a cantilever beam*
- *Spherical cavity in an infinite medium*
- *Performance of continuum and shell elements for linear analysis of bending problems*

Suggested reading

The volume of literature that has been written on the finite element method and the applications of finite element analysis is enormous. In most of the remaining chapters of this guide, a list of suggested books and articles is provided so that you can explore the topics in more depth if you wish. While the advanced references will not be of interest to most users, they provide detailed theoretical information for the interested user.

General texts on the finite element method

- NAFEMS Ltd., *A Finite Element Primer*, 1986.
- Becker, E. B., G. F. Carey, and J. T. Oden, *Finite Elements: An Introduction,* Prentice-Hall, 1981.
- Carey, G. F., and J. T. Oden, *Finite Elements: A Second Course,* Prentice-Hall, 1983.
- Cook, R. D., D. S. Malkus, and M. E. Plesha, *Concepts and Applications of Finite Element Analysis,* John Wiley & Sons, 1989.
- Hughes, T. J. R., *The Finite Element Method*, Prentice-Hall, Inc., 1987.
- Zienkiewicz, O. C., and R. L. Taylor, *The Finite Element Method: Volumes I, II, and III*, Butterworth-Heinemann, 2000.

Performance of linear solid elements

- Prathap, G., "The Poor Bending Response of the Four-Node Plane Stress Quadrilaterals," International Journal for Numerical Methods in Engineering, vol. 21, 825–835, 1985.

Hourglass control in solid elements

- Belytschko, T., W. K. Liu, and J. M. Kennedy, "Hourglass Control in Linear and Nonlinear Problems," Computer Methods in Applied Mechanics and Engineering, vol. 43, 251–276, 1984.
- Flanagan, D. P., and T. Belytschko, "A Uniform Strain Hexahedron and Quadrilateral with Hourglass Control," International Journal for Numerical Methods in Engineering, vol. 17, 679–706, 1981.
- Puso, M. A., "A Highly Efficient Enhanced Assumed Strain Physically Stabilized Hexahedral Element," International Journal for Numerical Methods in Engineering, vol. 49, 1029–1064, 2000.

Incompatible mode elements

- Simo, J. C. and M. S. Rifai, "A Class of Assumed Strain Methods and the Method of Incompatible Modes," International Journal for Numerical Methods in Engineering, vol. 29, 1595–1638, 1990.

Summary

This section presents a summary of the concepts.

- The formulation and order of integration used in a continuum element can have a significant effect on the accuracy and cost of the analysis.
- First-order (linear) elements using full integration are prone to shear locking and normally should not be used.
- First-order, reduced-integration elements are prone to hourglassing; sufficient mesh refinement minimizes this problem.
- When using first-order, reduced-integration elements in a simulation where bending deformation will occur, use at least four elements through the thickness.
- Hourglassing is rarely a problem in the second-order, reduced-integration elements in Abaqus/Standard. You should consider using these elements for most general applications when there is no contact.
- The accuracy of the incompatible mode elements available in Abaqus/Standard is strongly influenced by the amount of element distortion.
- The numerical accuracy of the results depends on the mesh that has been used. Ideally a mesh refinement study should be carried out to ensure that the mesh provides a unique solution to the problem. However, remember that using a converged mesh does not ensure that the results from the finite element simulation will match the actual behavior of the physical problem: that also depends on other approximations and idealizations in the model.
- In general, refine the mesh mainly in regions where you want accurate results; a finer mesh is required to predict accurate stresses than is needed to calculate accurate displacements.
- Advanced features such as submodeling are available in Abaqus to help you to obtain useful results for complex simulations.

Using Shell Elements

Use shell elements to model structures in which one dimension (the thickness) is significantly smaller than the other dimensions and in which the stresses in the thickness direction are negligible.
A structure, such as a pressure vessel, whose thickness is less than 1/10 of a typical global structural dimension generally can be modeled with shell elements. The following are examples of typical global dimensions:

- the distance between supports,
- the distance between stiffeners or large changes in section thickness,
- the radius of curvature, and
- the wavelength of the highest vibration mode of interest.

Abaqus shell elements assume that plane sections perpendicular to the shell midsurface remain plane. Do not be confused into thinking that the thickness must be less than 1/10 of the *element* dimensions. A highly refined mesh may contain shell elements whose thickness is greater than their in-plane dimensions, although this is not generally recommended—continuum elements may be more suitable in such a case.

In this section:

- *Element geometry*
- *Shell formulation – thick or thin*
- *Shell material directions*
- *Selecting shell elements*
- *Example: skew plate*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Element geometry

Two types of shell elements are available in Abaqus: conventional shell elements and continuum shell elements. Conventional shell elements discretize a reference surface by defining the element's planar dimensions, its surface normal, and its initial curvature. The nodes of a conventional shell element, however, do not define the shell thickness; the thickness is defined through section properties. Continuum shell elements, on the other hand, resemble three-dimensional solid elements in that they discretize an entire three-dimensional body yet are formulated so that their kinematic and constitutive behavior is similar to conventional shell elements. Continuum shell elements are more accurate in contact modeling than conventional shell elements, since they employ two-sided contact taking into account changes in thickness. For thin shell applications, however, conventional shell elements provide superior performance.

In this guide only conventional shell elements are discussed. Henceforth, we will refer to them simply as “shell elements.” For more information on continuum shell elements, see *About Shell Elements*.

In this section:

- *Shell thickness and section points*
- *Shell normals and shell surfaces*
- *Initial shell curvature*
- *Reference surface offsets*

Shell thickness and section points

The shell thickness is required to describe the shell cross-section and must be specified. In addition to specifying the shell thickness, you can choose to have the stiffness of the cross-section calculated during the analysis or once at the beginning of the analysis.

If you choose to have the stiffness calculated during the analysis, Abaqus uses numerical integration to calculate the stresses and strains independently at each section point (integration point) through the thickness of the shell, thus allowing nonlinear material behavior. For example, an elastic-plastic shell may yield at the outer section points while remaining elastic at the inner section points. The location of the single integration point in an S4R (4-node, reduced integration) element and the configuration of the section points through the shell thickness are shown in *Figure 1*.

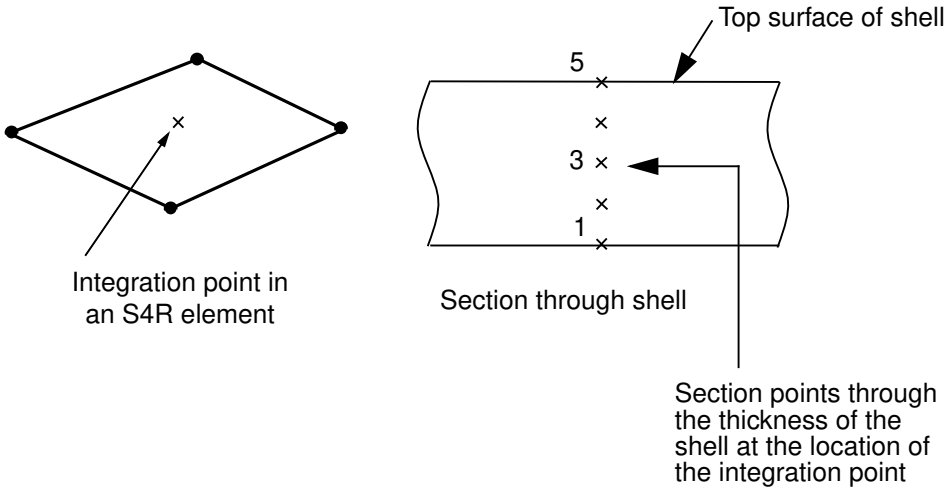


Figure 1: Configuration of section points in a numerically integrated shell.

You can specify any odd number of section points through the shell thickness when the properties are integrated during the analysis. By default, Abaqus uses five section points through the thickness of a homogeneous shell, which is sufficient for most nonlinear design problems. However, you should use more section points in some complicated simulations, especially when you anticipate reversed plastic bending (nine is normally sufficient in this case). For linear problems three section points provide exact integration through the thickness. However, calculating the material stiffness once at the beginning of the analysis is more efficient for linear elastic shells.

The material behavior must be linear elastic when the stiffness of the cross-section is calculated only at the beginning of the simulation. In this case all calculations are done in terms of the resultant forces and moments across the entire cross-section. If you request stress or strain output, Abaqus provides default output for the bottom surface, the midplane, and the top surface.

Shell normals and shell surfaces

The connectivity of the shell element defines the direction of the positive normal, as shown in *Figure 1*.

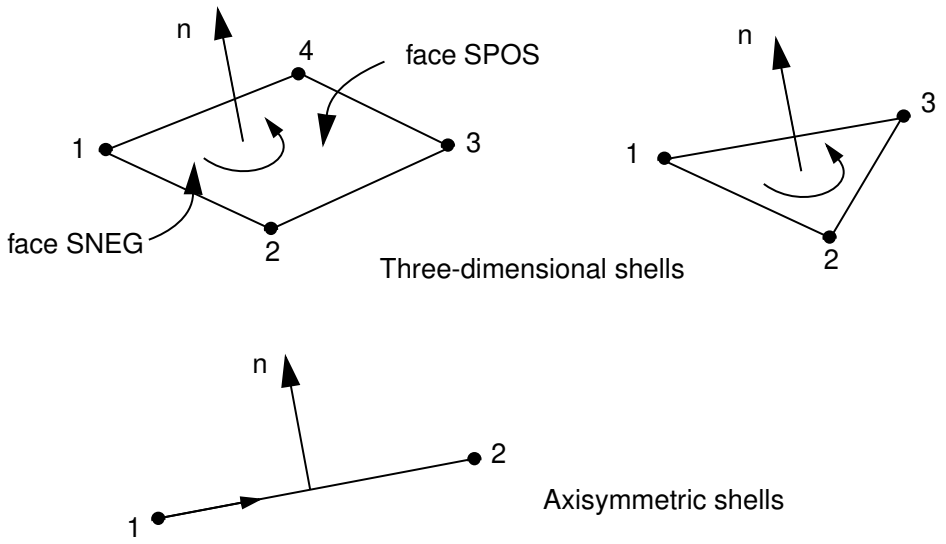


Figure 1: Positive normals for shells.

For axisymmetric shell elements the positive normal direction is defined by a 90° counterclockwise rotation from the direction going from node 1 to node 2. For three-dimensional shell elements the positive normal is given by the right-hand rule going around the nodes in the order in which they appear in the element definition.

The “top” surface of a shell is the surface in the positive normal direction and is called the SPOS face for contact definition. The “bottom” surface is in the negative direction along the normal and is called the SNEG face for contact definition. Normals should be consistent among adjoining shell elements.

The positive normal direction defines the convention for element-based pressure load application and output of quantities that vary through the shell thickness. A positive element-based pressure load applied to a shell element produces a load that acts in the direction of the positive normal. (The element-based pressure load convention for shell elements is opposite to that for continuum elements; the surface-based pressure load conventions for shell and continuum elements are identical. For more on the difference between element-based and surface-based distributed loads, see *Distributed Loads*.)

Initial shell curvature

Abaqus automatically calculates the surface normals at the nodes of every shell element to estimate the initial curvature of the shell. The surface normal at each node is determined using the algorithm discussed in *Defining the Initial Geometry of Conventional Shell Elements*.

With a coarse mesh as shown in *Figure 1*, Abaqus may determine several independent surface normals at the same node for adjoining elements. Physically, multiple normals at a single node mean that there is a fold line between the elements sharing the node. While it is possible that you intend to model such a structure, it is more likely that you intend to model a smoothly curved shell; Abaqus will try to smooth the shell by creating an averaged normal at a node.

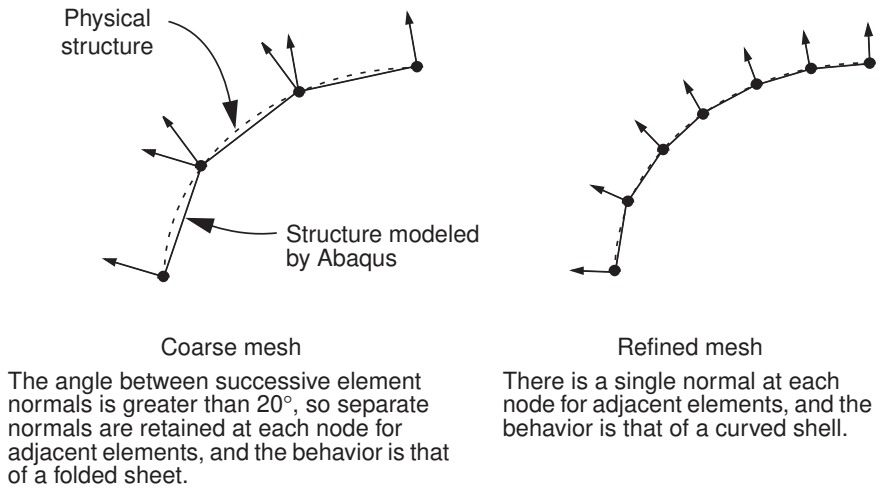


Figure 1: Effect of mesh refinement on the nodal surface normals.

The basic smoothing algorithm used is as follows: if the normals at a node for each shell element attached to the node are within 20° of each other, the normals will be averaged. The averaged normal will be used at that node for all elements attached to the node. If Abaqus cannot smooth the shell, a warning message is issued in the data (`.dat`) file.

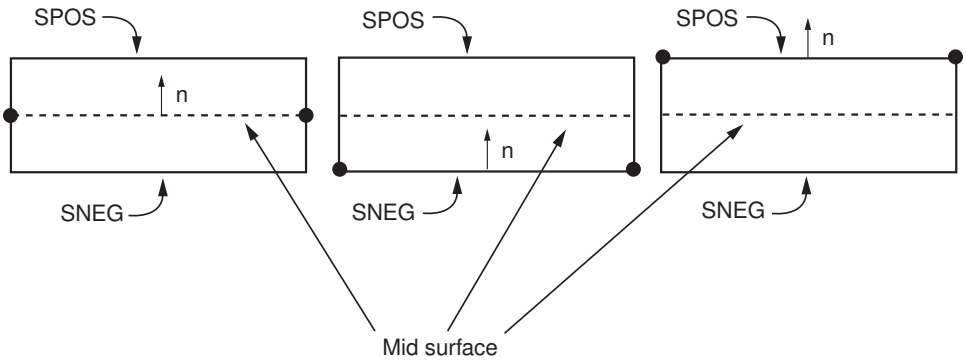
There are two methods that can be used to override the default algorithm. To introduce fold lines into a curved shell, either give the components of $\mathbf{n}$ as the 4th, 5th, and 6th data values following the nodal coordinates (this method requires manually editing the input file created by Abaqus/CAE in a text editor); or specify the normal direction directly with the **NORMAL* option (this option can be added using the Abaqus/CAE **Keywords Editor**; see *Cross-section orientation*). If both methods are used, the latter takes precedence. See *Defining the Initial Geometry of Conventional Shell Elements* for further details.

Reference surface offsets

The reference surface of the shell is defined by the shell element's nodes and normal definitions. When modeling with shell elements, the reference surface is typically coincident with the shell's midsurface. However, many situations arise in which it is more convenient to define the reference surface as offset from the shell's midsurface. For example, surfaces created in CAD packages usually represent either the top or bottom surface of the shell body. In this case it may be easier to define the reference surface to be coincident with the CAD surface and, therefore, offset from the shell's midsurface.

Shell offsets can also be used to define a more precise surface geometry for contact problems where shell thickness is important. Another situation where the offset from the midsurface may be important is when a shell with continuously varying thickness is modeled. In this case defining the nodes at the shell midplane can be difficult. If one surface is smooth while the other is rough, as in some aircraft structures, it is easiest to use shell offsets to define the nodes at the smooth surface.

Offsets can be introduced by specifying an offset value, which is defined as the distance (measured as a fraction of the shell's thickness) from the shell's midsurface to the reference surface containing the element's nodes. Positive values of the offset are in the positive normal direction. When the offset is set equal to 0.5 or SPOS, the top surface of the shell is the reference surface. When the offset is set equal to –0.5 or SNEG, the bottom surface is the reference surface. The default offset is 0, which indicates that the middle surface of the shell is the reference surface. These three reference surface offset settings are shown in *Figure 1* for a mesh where the nodal positions are adjusted to keep the position of the midsurface constant.



a) OFFSET= 0
Reference surface and midsurface are coincident

b) OFFSET= –0.5 (SNEG)
Reference surface is the bottom surface

c) OFFSET= +0.5 (SPOS)
Reference surface is the top surface

Figure 1: Schematic of shell offsets for offset values of 0, –0.5, and +0.5.

The degrees of freedom for the shell are associated with the reference surface. The element's area and all kinematic quantities are calculated there. Large offset values for curved shells may lead to a surface integration error, affecting the stiffness, mass, and rotary inertia for the shell section. For stability purposes Abaqus/Explicit also automatically augments the rotary inertia used for shell elements on the order of the offset squared, which may result in errors in the dynamics for large offsets. When large offsets from the shell's midsurface are necessary, use multi-point constraints or rigid body constraints instead.

Shell formulation – thick or thin

Shell problems generally fall into one of two categories: thin shell problems and thick shell problems. Thick shell problems assume that the effects of transverse shear deformation are important to the solution. Thin shell problems, on the other hand, assume that transverse shear deformation is small enough to be neglected. *Figure 1*(a) illustrates the transverse shear behavior of thin shells: material lines that are initially normal to the shell surface remain straight and normal throughout the deformation. Hence, transverse shear strains are assumed to vanish ($\gamma = 0$). *Figure 1*(b) illustrates the transverse shear behavior of thick shells: material lines that are initially normal to the shell surface do not necessarily remain normal to the surface throughout the deformation, thus adding transverse shear flexibility ($\gamma \neq 0$).

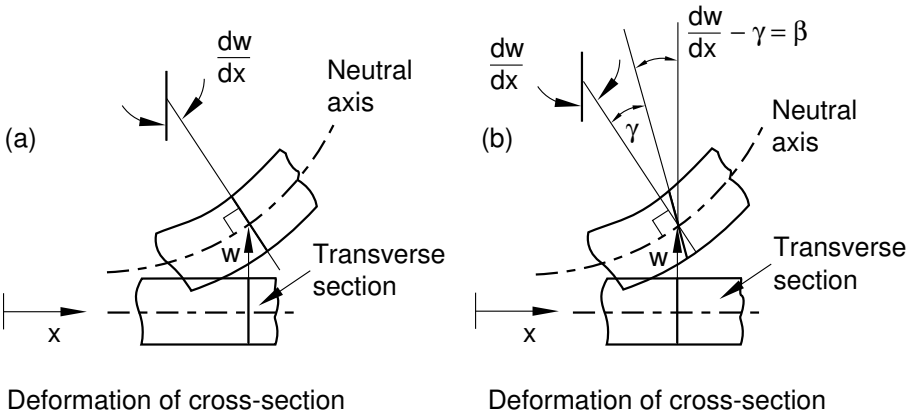


Figure 1: Behavior of transverse shell sections in (a) thin shells and (b) thick shells.

Abaqus offers multiple classes of shell elements, distinguished by the element's applicability to thin and thick shell problems. General-purpose shell elements are valid for use with both thick and thin shell problems. In certain cases, for specific applications, enhanced performance can be obtained by using the special-purpose shell elements available in Abaqus/Standard.

The special-purpose shell elements fall into two categories: thin-only shell elements and thick-only shell elements. All special-purpose shell elements provide for arbitrarily large rotations but only small strains. The thin-only shell elements enforce the Kirchhoff constraint; that is, plane sections normal to the midsection of the shell remain normal to the midsurface. The Kirchhoff constraint is enforced either analytically in the element formulation (STRI3) or numerically through the use of a penalty constraint. The thick-only shell elements are second-order quadrilaterals that may produce more accurate results than the general-purpose shell elements in small-strain applications where the loading is such that the solution is smoothly varying over the span of the shell.

To decide if a given application is a thin or thick shell problem, we can offer a few guidelines. For thick shells transverse shear flexibility is important, while for thin shells it is negligible. The significance of transverse shear in a shell can be estimated by its thickness-to-span ratio. A shell made of a single isotropic material with a ratio greater than 1/15 is considered “thick”; if the ratio is less than 1/15, the shell is considered “thin.” These estimates are approximate; you should always check the transverse shear effects in your model to verify the assumed shell behavior. Since transverse shear flexibility can be significant in laminated composite shell structures, this ratio should be much smaller for “thin” shell theory to apply. Composite shells with very compliant interior layers (so-called “sandwich” composites) have very low transverse shear stiffness and should almost always be modeled with “thick” shells; if the assumption of plane sections remaining plane is violated, continuum elements should be used. See *Shell Section Behavior* for details on checking the validity of using shell theory.

Transverse shear force and strain are available for general-purpose and thick-only shell elements. For three-dimensional elements, estimates of transverse shear stress are provided. The calculation of these stresses neglects coupling between bending and twisting deformation and assumes small spatial gradients of material properties and bending moments.

Shell material directions

Shell elements, unlike continuum elements, use material directions local to each element. Anisotropic material data, such as that for fiber-reinforced composites, and element output variables, such as stress and strain, are defined in terms of these local material directions. In large-displacement analyses the local material axes on a shell surface rotate with the average motion of the material at each integration point.

In this section:

- *Default local material directions*
- *Creating alternative material directions*

Default local material directions

The local material 1- and 2-directions lie in the plane of the shell. The default local 1-direction is the projection of the global 1-axis onto the shell surface. If the global 1-axis is normal to the shell surface, the local 1-direction is the projection of the global 3-axis onto the shell surface. The local 2-direction is perpendicular to the local 1-direction in the surface of the shell, so that the local 1-direction, local 2-direction, and the positive normal to the surface form a right-handed set (see *Figure 1*).

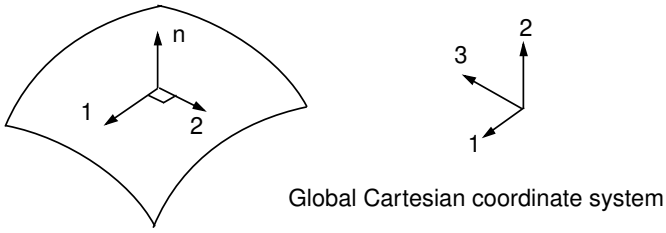


Figure 1: Default local shell material directions.

The default set of local material directions can sometimes cause problems; a case in point is the cylinder shown in *Figure 2*.

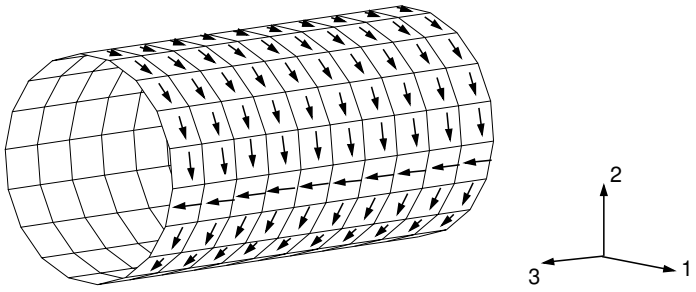


Figure 2: Default local material 1-direction in a cylinder.

For most of the elements in the figure the local 1-direction is circumferential. However, there is a line of elements that are normal to the global 1-axis. For these elements the local 1-direction is the projection of the global 3-axis onto the shell, making the local 1-direction axial instead of circumferential. A contour plot of the direct stress in the local 1-direction, σ_{11}, looks very strange, since for most elements σ_{11} is the circumferential stress, whereas for some elements it is the axial stress. In such cases it is necessary to define more appropriate local directions for the model, as discussed in the next section.

Creating alternative material directions

You can replace the global Cartesian coordinate system with a local rectangular, cylindrical, or spherical coordinate system.

Figure 1 shows local rectangular, cylindrical, and spherical coordinate systems definitions.

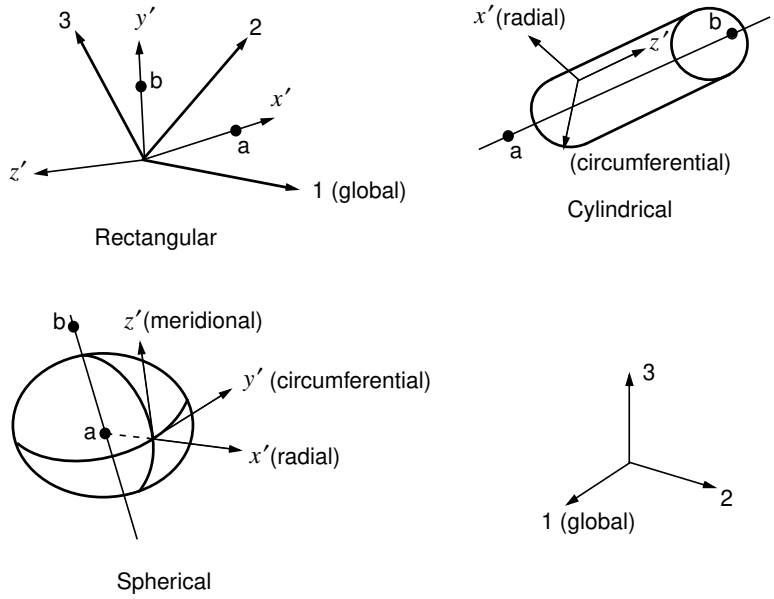


Figure 1: Definition of local coordinate systems.

You define the orientation of the local (x', y', z') coordinate system, as well as which of the local axes corresponds to which material direction. Thus, you must specify the local axis (1, 2, or 3) that is closest to being normal to the shell's surface. Abaqus follows a cyclic permutation (1, 2, 3) of the axes and projects the axis following your selection onto the shell region to form the material 1-direction. For example, if you choose the x'-axis, Abaqus projects the y'-axis onto the shell to form the material 1-direction. The material 2-direction is defined by the cross product of the shell normal and the material 1-direction. Normally, the final material 2-direction and the projection of the other local axis, in this case the z'-axis, will not coincide for curved shells.

If these local axes do not create the desired material directions, you can specify a rotation about the selected axis. The other two local axes are rotated by this amount before they are projected onto the shell's surface to give the final material directions. For the projections to be interpreted easily, the selected axis should be as close as possible to the shell normal.

For example, if the centerline of the cylinder shown in *Figure 2* coincides with the global 3-axis, local material directions can be defined such that the local material 1-direction is always circumferential and the corresponding local material 2-direction is always axial. The procedure is described below.

1. From the main menu bar of the Property module, select **Tools->Datum** and define a cylindrical datum coordinate system.
2. Select **Assign->Material Orientation** to assign a local material orientation to your part. When prompted to select a coordinate system, select the datum coordinate system defined in the previous step. The approximate shell normal direction is **Axis 1**; no additional rotation is necessary.

Selecting shell elements

Several guidelines are provided for selecting shell elements.

- The linear, finite-membrane-strain, fully integrated, quadrilateral shell element (S4) can be used when greater solution accuracy is desired, for problems prone to membrane- or bending-mode hourglassing, or for problems where in-plane bending is expected.
- The linear, finite-membrane-strain, reduced-integration, quadrilateral shell element (S4R) is robust and is suitable for a wide range of applications.
- The linear, finite-membrane-strain, triangular shell elements (S3/S3R) can be used as general-purpose elements. A refined mesh may be needed to capture bending deformations or high strain gradients because of the constant strain approximation in the elements.
- To account for the influence of shear flexibility in laminated composite shell models, use the shell elements suitable for modeling thick shells (S4, S4R, S3/S3R, S8R); check that the assumption of plane sections remaining plane is satisfied.
- Quadratic shell elements, either quadrilateral or triangular, are very effective for general, small-strain, thin-shell applications. These elements are not susceptible to shear or membrane locking.
- If you must use second-order elements in contact simulations, do not use the quadratic, triangular shell element (STRI65). Use the 9-node, quadrilateral shell element (S9R5) instead.
- For very large models that will experience only geometrically linear behavior, the linear, thin-shell element (S4R5) will generally be more cost-effective than the general-purpose shell elements.
- The small membrane strain elements are effective for explicit dynamics problems involving small membrane strains and arbitrarily large rotations.

Example: skew plate

This example discusses the modeling of a skew plate.

In this section:

- *Skew plate analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*

Skew plate analysis

This section discusses the skew plate example.

You have been asked to model the plate shown in *Figure 1*.

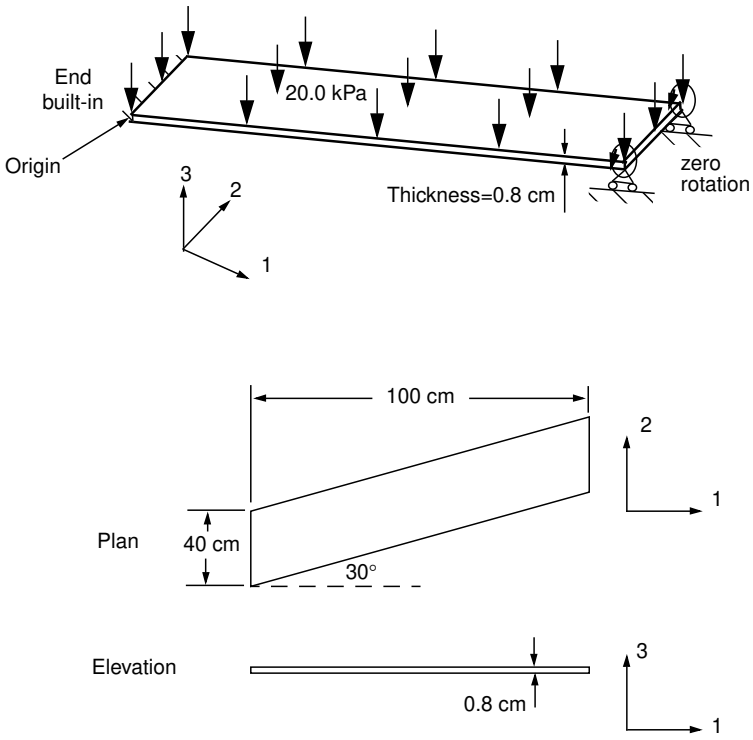


Figure 1: Sketch of the skew plate.

It is skewed 30° to the global 1-axis, is built-in at one end, and is constrained to move on rails parallel to the plate axis at the other end. You are to determine the midspan deflection when the plate carries a uniform pressure. You are also to assess whether a linear analysis is valid for this problem. You will perform an analysis using Abaqus/Standard.

Preprocessing—creating the model with Abaqus/CAE

Use Abaqus/CAE to create the entire model for this simulation.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Skew plate*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Skew plate**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Before you start to build the model, decide on a system of units. The dimensions are given in cm, but the loading and material properties are given in MPa and GPa. Since these are not consistent units, you must choose a consistent system to use in your model and convert the necessary input data. In the following discussion newtons, meters, kilograms, and seconds are used.

In this section:

- *Defining the model geometry*
- *Defining the material and section properties and the local material directions*
- *Creating an assembly, defining an analysis step, and specifying output requests*
- *Prescribing boundary conditions and applied loads*
- *Creating the mesh and defining a job*

Defining the model geometry

A suggested approach to creating the part geometry is outlined.

Start Abaqus/CAE, and create a three-dimensional, deformable body with a planar shell base feature. Name the part `Plate`, and specify an approximate part size of `4.0`.

1. In the Sketcher, create an arbitrary rectangle using the **Create Lines: Rectangle (4 Lines)** tool.
2. Delete all constraints automatically imposed by the sketcher (four perpendicular constraints and one horizontal constraint).
3. Constrain the left and right edges to remain vertical and the bottom and top edges to be parallel.
4. Dimension the left edge by selecting the line and assign it a value of `0.4` m.
5. Dimension the bottom edge. Select the vertices of the line rather than the line itself to define a horizontal dimension. Set the horizontal distance between the vertices to `1.0` m.

   Note:

   If you were to dimension the line itself, you would control only the length of the line (in whatever orientation it assumes).

6. Dimension the angle between the left and bottom edges. Select the left and bottom edges in succession, and set the value of the angle to `60°`.

   The final sketch is shown in *Figure 1*.

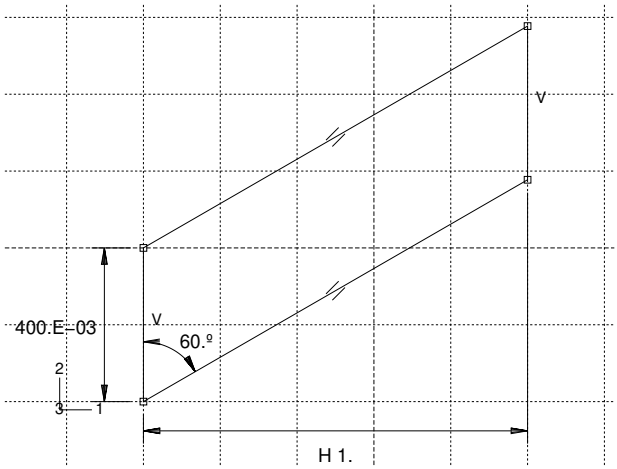


Figure 1: Sketch of the plate geometry (with grid spacing doubled).

7. In the prompt area, click **Done** to finish the sketch.

Defining the material and section properties and the local material directions

The plate is made of an isotropic, linear elastic material with a Young's modulus $E = 30 \times 10^9$ Pa and a Poisson's ratio $\nu = 0.3$.

Create the material definition; name the material `Metal`.

The orientation of the structure in the global coordinate system is shown in *Figure 1*. The global Cartesian coordinate system defines the default material directions, but the plate is skewed relative to this system. It will not be easy to interpret the results of the simulation if you use the default material directions because the direct stress in the material 1-direction, σ_{11}, will contain contributions from both the axial stress, produced by the bending of the plate, and the stress transverse to the axis of the plate. It will be easier to interpret the results if the material directions are aligned with the axis of the plate and the transverse direction. Therefore, a local rectangular coordinate system is needed in which the local x'-direction lies along the axis of the plate (i.e., at 30° to the global 1-axis) and the local y'-direction is also in the plane of the plate. Following the instructions given below, define the shell section properties in a local (nondefault) material coordinate system.

1. Define a homogeneous shell section named `PlateSection`. Specify that section integration be performed before the analysis since the material is linear elastic, and accept the default idealization option of **No idealization**. Assign a shell thickness of `0.8E-2` and the `Metal` material definition to the section.
2. Define a rectangular datum coordinate system as shown in *Figure 1* using the **Create Datum CSYS: 2 Lines** tool .

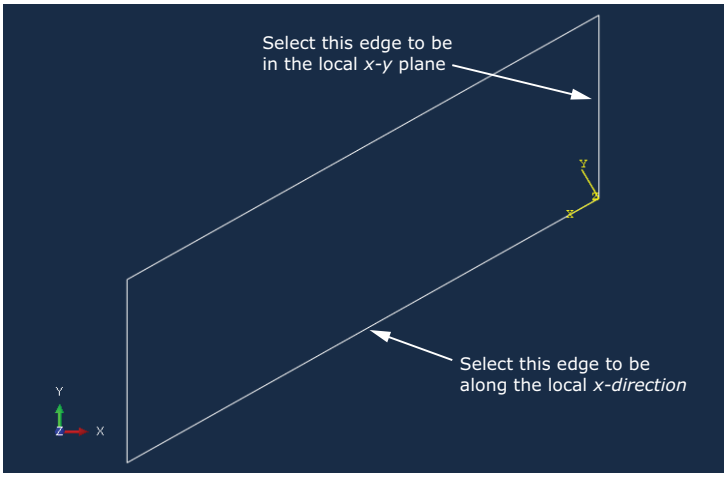


Figure 1: Datum coordinate system used to define local material directions.

3. From the main menu bar of the Property module, select **Assign->Material Orientation** and select the entire part as the region to which local material directions will be applied. In the viewport, select the datum coordinate system created earlier. Select **Axis 3** for the direction of the approximate shell normal. No additional rotation is needed about this axis.

   Tip: To verify that the local material directions have been assigned correctly, select **Tools->Query** from the main menu bar and perform a property query on the material orientations. Triads appear in the viewport indicating the material orientation of the region you select.

   Once the part has been meshed and elements have been created in the model, all element variables will be defined in this local coordinate system.

4. Assign the section definition to the plate. Accept **Middle surface** as the shell offset definition.

Creating an assembly, defining an analysis step, and specifying output requests

You will partition the plate in half at its midspan; this will allow you to define a set there. You will also define additional assembly-level sets to facilitate other output request and boundary condition definitions.

Create a dependent instance of the plate.

Next, create a single static, general step. Name the step `Apply Pressure`, and specify the following step description: `Uniform pressure (20 kPa) load`. Accept all the default settings for the step.

Among the output you will need are the nodal displacements, reaction forces, and element stresses as field data. These data will be used to create deformed shape plots, contour plots, and tabular data reports in the Visualization module. You will also want to write the displacements at the midspan as history data to create *X*–*Y* plots in the Visualization module.

Partition the plate and define geometry sets

1. In the Model Tree, double-click the **Plate** item in the **Parts** container to make it current.
2. Partition the plate in half using the **Partition Face: Use Shortest Path Between 2 Points** tool, . Use the midpoints of the skewed edges of the plate to create the partition shown in *Figure 1*.

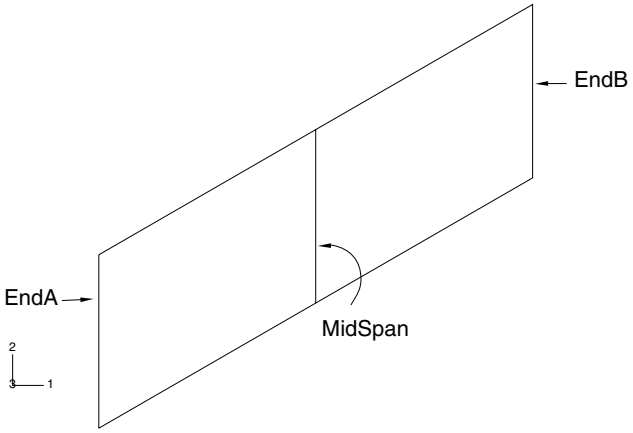


Figure 1: Partition used to define a geometry set at the plate midspan.

3. In the Model Tree, expand the **Assembly** container and double-click the **Sets** item to create a geometry set for the midspan named `MidSpan`. Similarly, create sets for the left and right edges of the plate and name them `EndA` and `EndB`, respectively. The locations of these three sets are indicated in *Figure 1*.



Tip: Geometry sets can be reviewed by expanding the **Sets** item underneath the **Assembly** container in the Model Tree and then double-clicking on the set name in the list that appears. The selected set is highlighted in the viewport, and its definition can be edited if necessary.

Change the default output requests

1. Edit the field output request so that only the nodal displacements, reaction forces, and element stresses and strains for the whole model are written as field data to the `.odb` file.
2. Edit the history output request so that only the vertical nodal displacements **U3** for the set named `MidSpan` are written as history data to the `.odb` file.

Prescribing boundary conditions and applied loads

As shown in *Figure 1*, the left end of the plate is completely fixed; the right end is constrained to move on rails that are parallel to the axis of the plate. Since the latter boundary condition direction does not coincide with the global axes, you must define a local coordinate system that has an axis aligned with the plate. You can use the datum coordinate system that you created earlier to define the local material directions.

1. In the Model Tree, double-click the **BCs** container and define a **Displacement/Rotation** mechanical boundary condition named `Rail boundary condition` in the `Apply Pressure` step.

   In this example you will assign boundary conditions to sets rather than to regions selected directly in the viewport. Thus, when prompted for the regions to which the boundary condition will be applied, click **Sets** in the prompt area of the viewport.

2. From the **Region Selection** dialog box that appears, select set **EndB**. Toggle on **Highlight selections in viewport** to make sure the correct set is selected. The right edge of the plate should be highlighted. Click **Continue**.

3. In the **Edit Boundary Condition** dialog box, click  to specify the local coordinate system in which the boundary condition will be applied. In the viewport, select the datum coordinate system that was created earlier to define the local directions. The local 1-direction is aligned with the plate axis.

4. In the **Edit Boundary Condition** dialog box, fix all degrees of freedom except for **U1**.

   The right edge of the plate is now constrained to move only in the direction of the plate axis. Once the plate has been meshed and nodes have been generated in the model, all printed nodal output quantities associated with this region (displacements, velocities, reaction forces, etc.) will be defined in this local coordinate system.

Complete the boundary condition definition by fixing all degrees of freedom at the left edge of the plate (set **EndA**). Name this boundary condition `Fix left end`. Use the default global directions for this boundary condition.

Finally, define a uniform pressure load across the top of the shell named `Pressure`. Select both regions of the part using **[Shift] + Click** and then click **Done**. When prompted to choose a side for the shell or internal faces, select **Brown**, which corresponds to the top side of the plate. You may need to rotate the view to more clearly distinguish the top side of the plate. Specify a load magnitude of `2.E4` Pa.

Creating the mesh and defining a job

Figure 1 shows the suggested mesh for this simulation.

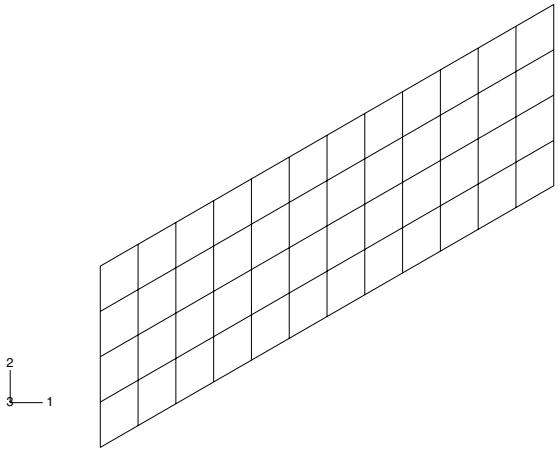


Figure 1: Suggested mesh design for the skew plate simulation.

You must answer the following questions before selecting an element type: Is the plate thin or thick? Are the strains small or large? The plate is quite thin, with a thickness-to-minimum span ratio of 0.02. (The thickness is 0.8 cm and the minimum span is 40 cm.) While we cannot readily predict the magnitude of the strains in the structure, we think that the strains will be small. Based on this information, choose quadratic shell elements (S8R5) because they give accurate results for thin shells in small-strain simulations. For further details on shell element selection, refer to *Choosing a Shell Element*.

In the Model Tree, expand the **Plate** item underneath the **Parts** container and double-click **Mesh** in the list that appears. Seed the part using a global element size of `0.1`. Create a quadrilateral mesh using quadratic, reduced-integration shell elements with five degrees of freedom per node (S8R5).

Before proceeding, rename the model to `Linear`. This model will later form the basis of the model used in the skew plate example discussed in *Nonlinearity*.

Define a job named `SkewPlate` with the following description:

```
Linear Elastic Skew Plate. 20 kPa Load.
```

Save your model in a model database file named `SkewPlate.cae`.

Submit the job for analysis, and monitor the solution progress; correct any modeling errors detected by the analysis product, and investigate the cause of any warnings.

Postprocessing

This section discusses postprocessing with Abaqus/CAE. Both contour and symbol plots are useful for visualizing shell analysis results.
Since contour plotting was discussed in detail in *Using Continuum Elements*, we use symbol plots here.

In the **Module** list located in the context bar, click **Visualization** to enter the Visualization module. Then, open the `.odb` file created by this job (`SkewPlate.odb`).

By default, Abaqus/CAE plots the undeformed shape of the model.

In this section:

- *Element normals*
- *Symbol plots*
- *Material directions*
- *Evaluating results based on tabular data*

Element normals

Use the undeformed shape plot to check the model definition.
Check that the element normals for the skew-plate model were defined correctly and point in the positive 3-direction.

Display the element normals

1. From the main menu bar, select **Options->Common**; or use the tool in the toolbox.
   The **Common Plot Options** dialog box appears.
2. Click the **Normals** tab.
3. Toggle on **Show normals**, and accept the default setting of **On elements**.
4. Click **OK** to apply the settings and to close the dialog box.
   The default view is isometric. You can change the view using the options in the view menu or the view tools (such as) from the View Manipulation toolbar.

Change the view

1. From the main menu bar, select **View->Specify**.
   The **Specify View** dialog box appears.
2. From the list of available methods, select **Viewpoint**.
3. Enter the ***X***-, ***Y***- and ***Z***-coordinates of the viewpoint vector as `-0.2, -1, 0.8` and the coordinates of the up vector as `0, 0, 1`.
4. Click **OK**.
   Abaqus/CAE displays your model in the specified view, as shown in *Figure 1*.

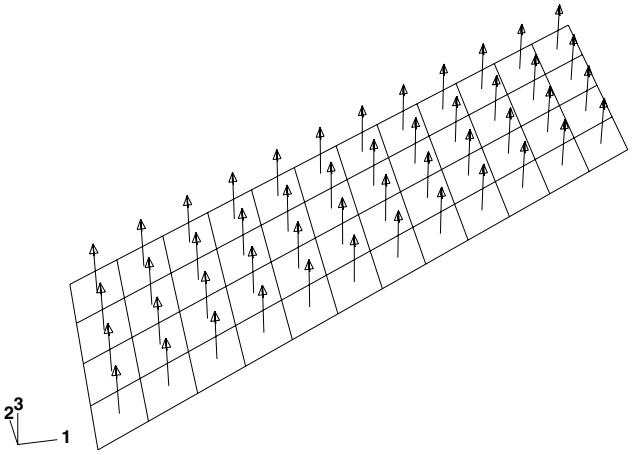


Figure 1: Shell element normals in the skew plate model.

Symbol plots

Symbol plots display the specified variable as a vector originating from the node or element integration points. You can produce symbol plots of most tensor- and vector-valued variables. The exceptions are mainly nonmechanical output variables and element results stored at nodes, such as nodal forces. The relative size of the arrows indicates the relative magnitude of the results, and the vectors are oriented along the global direction of the results. The symbol plot legend shows how each arrow color corresponds to a specific range of values. You can plot results for the resultant of variables such as displacement (U), reaction force (RF), etc.; or you can plot individual components of these variables.

Before proceeding, suppress the visibility of the element normals.

Generate a symbol plot of the displacement

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Symbol**.
2. From the list of output variables in the center of the toolbar, select **U**.
3. From the list of vector quantities and selected components, select **U3**.

   Abaqus/CAE displays a symbol plot of the displacement vector resultant on the deformed model shape, as shown in *Figure 1*.
4. The default shaded render style obscures the arrows. An unobstructed view of the arrows can be obtained by changing the render style to **Wireframe** using the **Common Plot Options** dialog box. If the element normals are still visible, you should turn them off at this time.

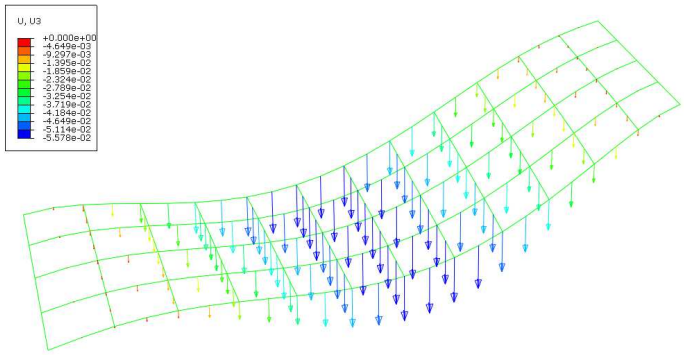


Figure 1: Symbol plot of displacement.

Generate a symbol plot of the principal stresses

You can plot principal values of tensor variables such as stress using symbol plots. A symbol plot of the principal values of stress yields three vectors at every integration point, each corresponding to a principal value oriented along the corresponding principal direction. Compressive values are indicated by arrows pointing toward the integration point, and tensile values are indicated by arrows pointing away from the integration point. You can also plot individual principal values.

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Symbol**.
2. From the list of output variables in the center of the toolbar, select **S**.
3. From the list of tensor quantities and components, select **ALL_PRINCIPAL_COMPONENTS** as the tensor quantity.

   Abaqus/CAE displays a symbol plot of principal stresses.
4. From the main menu bar, select **Options->Symbol**; or use the **Symbol Options** tool in the toolbox to change the arrow length.

The **Symbol Plot Options** dialog box appears.

5. In the **Color & Style** page, click the **Tensor** tab.
6. Drag the **Size** slider to select 2 as the arrow length.
7. Click **OK** to apply the settings and to close the dialog box.

   The symbol plot shown in *Figure 2* appears.

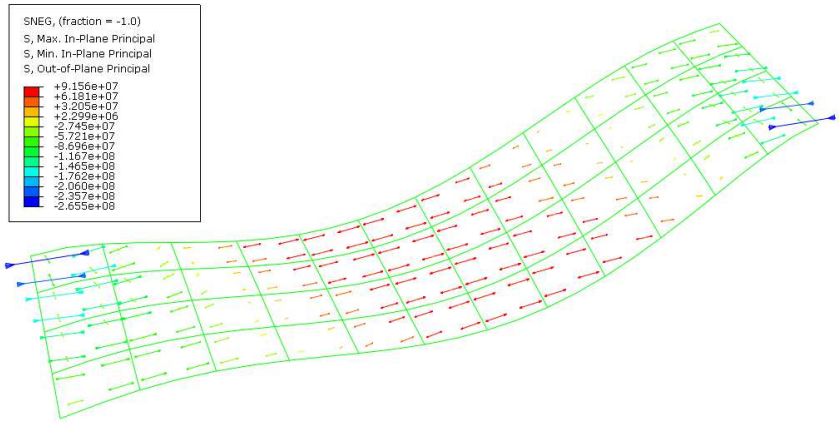


Figure 2: Symbol plot of principal stresses on the bottom surface of the plate.

8. The principal stresses are displayed at section point 1 by default. To plot stresses at nondefault section points, select **Result->Section Points** from the main menu bar to open the **Section Points** dialog box.
9. Select the desired nondefault section point for plotting.
10. In a complex model, the element edges can obscure the symbol plots. To suppress the display of the element edges, choose **Feature edges** in the **Basic** tabbed page of the **Common Plot Options** dialog box.

    Figure 3 shows a symbol plot of the principal stresses at the default section point with only feature edges visible.

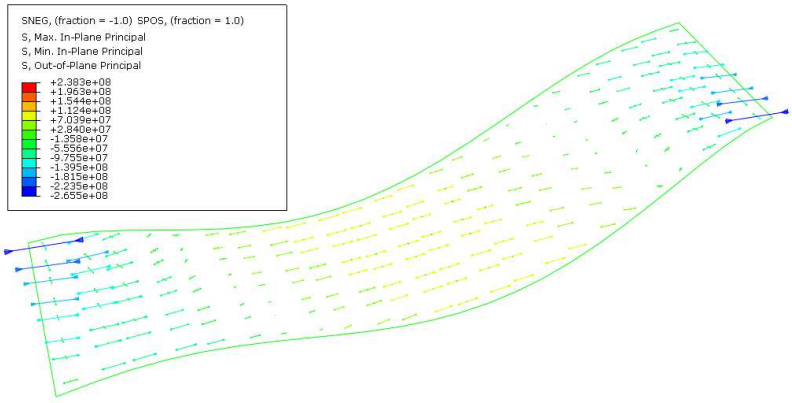


Figure 3: Symbol plot of principal stresses on the top and bottom surfaces of the plate using feature edges.

Material directions

Abaqus/CAE also allows you visualize the element material directions. This feature is particularly useful if you would like to verify that the material directions were assigned correctly in the simulation.

1. From the main menu bar, select **Plot->Material Orientations->On Undeformed Shape**; or use the  tool in the toolbox.

   The material orientation directions are plotted on the undeformed shape. By default, the triads that represent the material orientation directions are plotted without arrowheads.

2. From the main menu bar, select **Options->Material Orientation**; or use the **Material Orientation Options**  tool in the toolbox to display the triads with arrowheads.

   The **Material Orientation Plot Options** dialog box appears.

3. Set the **Arrowhead** option to use filled arrowheads in the triad.
4. Click **OK** to apply the settings and to close the dialog box.
5. Use the predefined views available in the Views toolbar to display the plate as shown in *Figure 1*. In this figure, perspective is turned off. To turn off perspective, click the  tool in the View Options toolbar.

    **Tip:** If the Views toolbar is not visible, select **View->Toolbars->Views** from the main menu bar.

   By default, the material 1-direction is colored blue, the material 2-direction is colored yellow, and, if it is present, the material 3-direction is colored red.

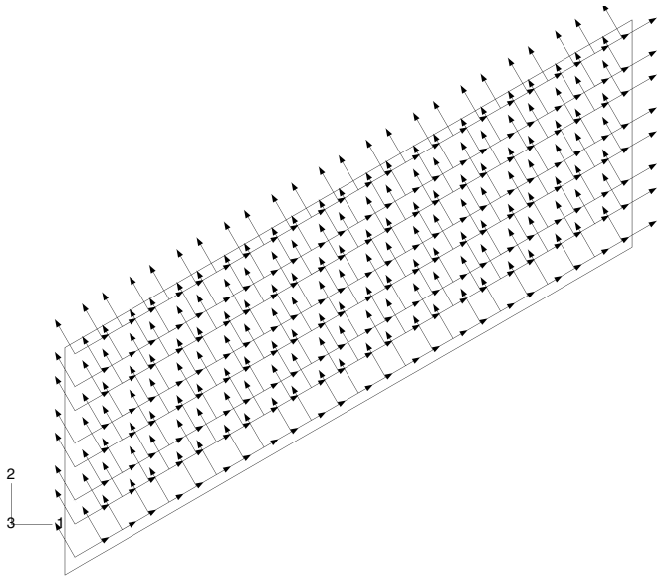


Figure 1: Plot of material orientation directions in the plate.

Evaluating results based on tabular data

Additional postprocessing can be performed by examining printed data.

With the aid of display groups, create a tabular data report of the whole model element stresses (components **S11**, **S22**, and **S12**), the reaction forces and moments at the supported nodes (sets **EndA** and **EndB**), and the displacements of the midspan nodes (set **MidSpan**). The stress data are shown below.

```
Field Output Report

Source 1
---------

   ODB: SkewPlate.odb
   Step: Apply pressure
   Frame: Increment      1: Step Time =    1.000

Loc 1 : Integration point values at shell general ... : SNEG, (fraction = -1.0)
Loc 2 : Integration point values at shell general ... : SPOS, (fraction =  1.0)

Output sorted by column "Element Label".

Field Output reported at integration points for part: PLATE-1

  Element    Int          S.S11         S.S11         S.S22         S.S22
    S.S12         S.S12
    Label     Pt         @Loc 1        @Loc 2        @Loc 1        @Loc 2
   @Loc 1        @Loc 2
------------------------------------------------------------------------------


        1      1   79.7614E+06  -79.7614E+06    1.1085E+06   -1.1085E+06
-5.86291E+06   5.86291E+06
        1      2   83.7703E+06  -83.7703E+06   7.14559E+06  -7.14559E+06
-8.00706E+06   8.00706E+06
        1      3   66.9385E+06  -66.9385E+06   2.79241E+06  -2.79241E+06
-1.98396E+06   1.98396E+06
        1      4   72.3479E+06  -72.3479E+06   5.05957E+06  -5.05957E+06
-7.0819E+06    7.0819E+06
        .
        .
       48      1  -142.755E+06   142.755E+06  -56.0747E+06   56.0747E+06
21.007E+06   -21.007E+06
       48      2  -118.848E+06   118.848E+06  -7.21449E+06   7.21449E+06
4.00065E+06  -4.00065E+06
       48      3   -187.19E+06    187.19E+06   -103.31E+06    103.31E+06
50.352E+06   -50.352E+06
       48      4  -238.323E+06   238.323E+06  -84.7331E+06   84.7331E+06
70.0676E+06  -70.0676E+06


  Minimum         -238.323E+06  -90.2378E+06   -103.31E+06  -10.5216E+06
-18.865E+06  -70.0676E+06
      At Element            48             4            24             2
      12            48

          Int Pt             4             4             3             2
       4             4
  Maximum          90.2378E+06   238.323E+06   10.5216E+06    103.31E+06
70.0676E+06    18.865E+06
      At Element             4            48             2            24
```

```
      48            12

          Int Pt             4             4             2             3
       4             4
```

The locations `Loc 1` and `Loc 2` identify the section point in the element where the stress was calculated. `Loc 1` (corresponding to section point 1) lies on the SNEG surface of the shell, and `Loc 2` (corresponding to section point 3) lies on the SPOS surface. Local material directions have been used for the element: the stresses refer to a local coordinate system.

Check that the small-strain assumption was valid for this simulation. The axial strain corresponding to the peak stress is $\varepsilon_{11} \approx 0.008$. Because the strain is typically considered small if it is less than 4 or 5%, a strain of 0.8% is well within the appropriate range to be modeled with S8R5 elements.

The reaction forces and moments are reported in the following table:

```
Field Output Report

Source 1
---------

   ODB: SkewPlate.odb
   Step: Apply pressure
   Frame: Increment      1: Step Time =    1.000

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".

Field Output reported at nodes for part: PLATE-1

   Node       RF.RF1       RF.RF2       RF.RF3       RM.RM1       RM.RM2
 RM.RM3
  Label       @Loc 1       @Loc 1       @Loc 1       @Loc 1       @Loc 1
 @Loc 1
-------------------------------------------------------------------------------

      3           0.           0.      37.3924      -1.5991     -76.4939
     0.
      4           0.           0.     -109.834      1.77236 -324.424E-03
     0.
      5           0.           0.      37.3918      1.59909      76.4939
     0.
      6           0.           0.     -109.834     -1.77236  324.411E-03
     0.
     15           0.           0.      73.6366      8.75019     -62.2243
     0.
     16           0.           0.      260.424      6.95105     -51.1181
     0.
     17           0.           0.      239.685      6.56987     -35.4374
     0.
     28           0.           0.      73.6366     -8.75019      62.2243
     0.
     29           0.           0.      260.424     -6.95105      51.1181
     0.
     30           0.           0.      239.685     -6.56988      35.4374
     0.
    116           0.           0.      6.15382       7.5915     -36.4274
     0.
    119           0.           0.      455.132      6.80781      -88.237
     0.
```

```
   121            0.            0.      750.805      8.31069     -126.462
    0.
   123            0.            0.   2.2866E+03      31.0977     -205.818
    0.
   170            0.            0.        6.154      -7.5915      36.4274
    0.
   173            0.            0.      455.133     -6.80782       88.237
    0.
   175            0.            0.      750.805     -8.31069      126.462
    0.
   177            0.            0.   2.2866E+03     -31.0977      205.818
    0.


 Minimum          0.            0.     -109.834     -31.0977     -205.818
    0.
     At Node    177           177            6          177           123
  177

 Maximum          0.            0.   2.2866E+03      31.0977      205.818
    0.
     At Node    177           177          177          123           177
  177

       Total      0.            0.  8.00000E+03 -39.2199E-06 -5.00679E-06
    0.
```

The reaction forces are written in the global coordinate system. Check that the sum of the reaction forces and reaction moments with the corresponding applied loads is zero. The nonzero reaction force in the 3-direction equilibrates the vertical force of the pressure load (20 kPa × 1.0 m × 0.4 m). In addition to the reaction forces, the pressure load causes self-equilibrating reaction moments at the constrained rotational degrees of freedom.

This example was run as a linear analysis, in which it is assumed that the nodal displacements are small relative to the characteristic structural dimensions. The midspan deflection across the plate, as indicated in the table of displacements (not shown here), is approximately 5.4 cm, or roughly 5% of the plate's length. However, it is questionable whether the displacements are sufficiently small for a linear analysis to provide accurate results. Nonlinear effects in the structure may be important, so we need to run a nonlinear analysis to further investigate this example. Geometrically nonlinear analyses are discussed in *Nonlinearity*.

Related Abaqus examples

Related Abaqus examples are available.

- *Pressurized fuel tank with variable shell thickness*
- *Analysis of an anisotropic layered plate*
- *Buckling of a simply supported square plate*
- *The barrel vault roof problem*

Suggested reading

The following references provide a more in-depth treatment of the theoretical and computational aspects of shell theory.

Basic shell theory

- Timoshenko, S., *Strength of Materials: Part II*, Krieger Publishing Co., 1958.
- Timoshenko, S. and S. W. Krieger, *Theory of Plates and Shells*, McGraw-Hill, Inc., 1959.
- Ugural, A. C., *Stresses in Plates and Shells*, McGraw-Hill, Inc., 1981.

Basic computational shell theory

- Cook, R. D., D. S. Malkus, and M. E. Plesha, *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, 1989.
- Hughes, T. J. R., *The Finite Element Method*, Prentice-Hall, Inc., 1987.

Advanced shell theory

- Budiansky, B., and J. L. Sanders, “On the `Best' First-Order Linear Shell Theory,” Progress in Applied Mechanics, The Prager Anniversary Volume, 129–140, 1963.

Advanced computational shell theory

- Ashwell, D. G., and R. H. Gallagher, *Finite Elements for Thin Shells and Curved Members*, John Wiley & Sons, 1976.
- Hughes, T. J. R., T. E. Tezduyar, “Finite Elements Based upon Mindlin Plate Theory with Particular Reference to the Four-Node Bilinear Isoparametric Element,” Journal of Applied Mechanics, 587–596, 1981.
- Simo, J. C., D. D. Fox, and M. S. Rifai, “On a Stress Resultant Geometrically Exact Shell Model. Part III: Computational Aspects of the Nonlinear Theory,” Computer Methods in Applied Mechanics and Engineering, vol. 79, 21–70, 1990.

Summary

This section presents a summary of the concepts.

- The cross-section behavior of shell elements can be determined using numerical integration through the shell thickness or using a cross-section stiffness calculated at the beginning of the analysis.
- Calculating the cross-section stiffness at the beginning of the analysis is efficient, but only linear materials can be considered when this is done; calculating the cross-section stiffness during the analysis using numerical integration allows both linear and nonlinear materials to be used.
- Numerical integration is performed at a number of section points through the shell thickness. These section points are the locations at which element variables can be output. The default outermost section points lie on the surfaces of the shell.
- The direction of a shell element's normal determines the positive and negative surfaces of the element. To define contact and interpret element output correctly, you must know which surface is which. The shell normal can be plotted in the Visualization module of Abaqus/CAE.
- Shell elements use material directions local to each element. In large-displacement analyses the local material axes rotate with the element. Nondefault local coordinate systems can be defined. The element variables, such as stress and strain, are output in the local directions.
- Local coordinate systems for nodes can also be defined. Concentrated loads and boundary conditions are applied in the local coordinate system. All printed nodal output, such as displacements, also refer to the local system by default.
- Symbol plots can help you visualize the results from a simulation. They are especially useful for visualizing the motion and load paths of a structure.

Using Beam Elements

Use beam elements to model structures in which one dimension (the length) is significantly greater than the other two dimensions and in which the longitudinal stress is most important.
Beam theory is based on the assumption that the deformation of the structure can be determined entirely from variables that are functions of position along the structure's length. For beam theory to produce acceptable results, the cross-section dimensions should be less than 1/10 of the structure's typical axial dimension. The following are examples of typical axial dimensions:

- the distance between supports,
- the distance between gross changes in cross-section, and
- the wavelength of the highest vibration mode of interest.

Abaqus beam elements assume that plane sections perpendicular to the axis of the beam remain plane during deformation.

Do not be confused into thinking that the cross-section dimensions should be less than 1/10 of a typical *element* length. A highly refined mesh may contain beam elements whose length is less than their cross-section dimensions, although this is not generally recommended—continuum elements may be more suitable in such a case.

In this section:

- *Beam cross-section geometry*
- *Formulation and integration*
- *Selecting beam elements*
- *Example: cargo crane*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Beam cross-section geometry

This section discusses the definition of beam cross-section geometry.

In this section:

- *Beam cross-section analysis*
- *Section points*
- *Cross-section orientation*
- *Beam element curvature*
- *Nodal offsets in beam sections*

Beam cross-section analysis

You can define the beam profile in one of three ways: by choosing from the Abaqus cross-section library and specifying the shape and dimensions of the beam cross-section; by defining a generalized beam profile using section engineering properties, such as area and moments of inertia; or by using a mesh of special two-dimensional elements for which geometric quantities are calculated numerically, called a meshed beam cross-section.

Abaqus offers a variety of common cross-section shapes, as shown in *Figure 1*, should you decide to define the beam profile geometrically. You can also define almost any thin-walled cross-section using the arbitrary cross-section definition. For a detailed discussion of the beam cross-sections available in Abaqus, see *Beam Cross-Section Library*.

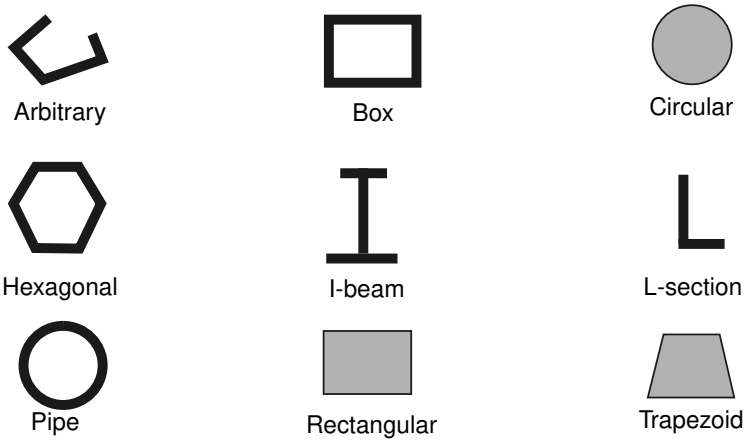


Figure 1: Beam cross-sections.

If you define a beam profile using one of the built-in cross-sections in the Abaqus library, Abaqus/CAE prompts you for the required cross-section dimensions, which are different for each type of cross-section. When the beam profile is associated with a beam section property, you can specify whether to have the section engineering properties calculated during the analysis or to have Abaqus precompute them (at the beginning of the analysis). The former option can be used when the material behavior is either linear or nonlinear (for example, if the section stiffness changes due to inelastic yielding); the latter option is more efficient, however, for linear elastic material behavior.

Alternatively, you can provide the section engineering properties (area, moments of inertia, and torsional constants) instead of the cross-section dimensions. The material behavior may be either linear or nonlinear. Thus, you can combine the beam's geometry and material behavior to define its response to loads, which may be linear or nonlinear. See *Using a General Beam Section to Define the Section Behavior* for further details.

In Abaqus/Standard you can also define beams with linearly tapered cross-sections. General beam sections with linear response and standard library sections are supported.

Meshed beam cross-sections allow a description of the beam cross-section that includes multiple materials and complex geometry. This type of beam profile is discussed further in *Meshed Beam Cross-Sections*.

Section points

When you define the beam cross-section using a built-in profile from the Abaqus cross-section library and request that section engineering properties be calculated during the analysis, Abaqus calculates the beam element's response at an array of section points throughout the beam cross-section. The number of section points, as well as the section point locations, are shown in *Beam Cross-Section Library*. Element output variables, such as stress and strain, are available at any of the section points; however, by default output is provided at only a select number of section points, as described in *Beam Cross-Section Library*. All the section points for a rectangular cross-section are shown in *Figure 1*.

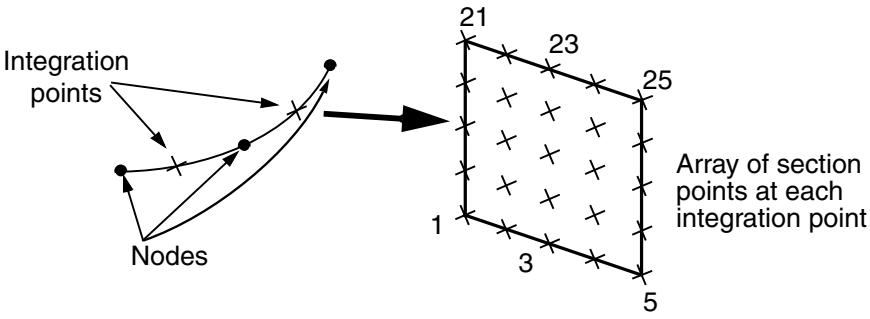


Figure 1: Integration and default section points in a B32 rectangular beam element.

For this cross-section output is provided at points 1, 5, 21, and 25 by default. The beam element shown in *Figure 1* uses a total of 50 section points, 25 at each of the two integration points, to calculate its stiffness.

When you request that the beam section properties be precomputed, Abaqus does not calculate the beam's response at the section points. Instead, it uses the section engineering properties to determine the section response. Therefore, Abaqus uses section points only as locations for output, and you need to specify the section points at which you desire output.

Cross-section orientation

You must define the orientation of a beam's cross-section in global Cartesian space. The local tangent along the beam element, $\mathbf{t}$, is defined as the vector along the element axis pointing from the first node of the element to the next node. The beam cross-section is perpendicular to this local tangent. The local (1–2) beam section axes are represented by the vectors $\mathbf{n_1}$ and $\mathbf{n_2}$. The three vectors $\mathbf{t}$, $\mathbf{n_1}$, and $\mathbf{n_2}$ form a local, right-handed, coordinate system (see *Figure 1*).

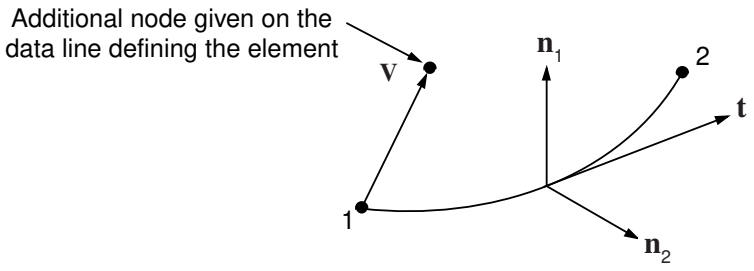


Figure 1: Orientation of the beam element tangent, $\mathbf{t}$, and beam section axes, $\mathbf{n_1}$ and $\mathbf{n_2}$.

The $\mathbf{n_1}$-direction is always (0.0, 0.0, −1.0) for two-dimensional beam elements.

For three-dimensional beam elements there are several ways to define the orientation of the local beam section axes. The first is to specify an extra node on the data line defining the element (this method requires manually editing the input file created by Abaqus/CAE). The vector, $\mathbf{v}$, from the first node in the beam element to this additional node (see *Figure 1*), is used initially as an approximate $\mathbf{n_1}$-direction. Abaqus then defines the beam's $\mathbf{n_2}$-direction as $\mathbf{t} \times \mathbf{v}$. Having determined $\mathbf{n_2}$, Abaqus defines the actual $\mathbf{n_1}$-direction as $\mathbf{n_2} \times \mathbf{t}$. This procedure ensures that the local tangent and local beam section axes form an orthogonal system.

Alternatively, you can give an approximate $\mathbf{n_1}$-direction when you define the beam section properties in Abaqus/CAE. Abaqus then uses the procedure described above to calculate the actual beam section axes. If you specify both an extra node and an approximate $\mathbf{n_1}$-direction, the additional node method takes precedence. Abaqus uses the vector from the origin to the point (0.0, 0.0, −1.0) as the default $\mathbf{n_1}$-direction if you provide no approximate $\mathbf{n_1}$-direction.

There are two methods that can be used to override the $\mathbf{n_2}$-direction defined by Abaqus; both require editing the input file manually. One is to give the components of $\mathbf{n_2}$ as the 4th, 5th, and 6th data values following the nodal coordinates. The alternative is to specify the normal direction directly with the **NORMAL* option (this option can be added using the Abaqus/CAE**Keywords Editor**). If both methods are used, the latter takes precedence. Abaqus again defines the $\mathbf{n_1}$-direction as $\mathbf{n_2} \times \mathbf{t}$.

The $\mathbf{n_2}$-direction that you provide need not be orthogonal to the beam element tangent, $\mathbf{t}$. When you provide the $\mathbf{n_2}$-direction, the local beam element tangent is redefined as the value of the cross product $\mathbf{n_1} \times \mathbf{n_2}$. It is quite possible in this situation that the redefined local beam tangent, $\mathbf{t}$, will not align with the beam axis, as defined by the vector from the first to the second node. If the $\mathbf{n_2}$-direction subtends an angle greater than 20° with the plane perpendicular to the element axis, Abaqus issues a warning message in the data file.

The example presented in *Example: cargo crane* explains how to assign the beam cross-section orientation using Abaqus/CAE.

Beam element curvature

The curvature of beam elements is based on the orientation of the beam's $\mathbf{n_2}$-direction relative to the beam axis. If the $\mathbf{n_2}$-direction and the beam axis are not orthogonal (i.e., the beam axis and the tangent, $\mathbf{t}$, do not coincide), the beam element is considered to be curved initially. Since the behavior of curved beams is different from the behavior of straight beams, you should always check your model to ensure that the correct normals and, hence, the correct curvatures are used. For beams and shells Abaqus uses the same algorithm to determine the normals at nodes shared by several elements. A description is given in *Beam Element Cross-Section Orientation*.

If you intend to model curved beam structures, you should use one of the two methods described earlier to define the $\mathbf{n_2}$-direction directly, allowing you great control in modeling the curvature. Even if you intend to model a structure made up of straight beams, curvature may be introduced as normals are averaged at shared nodes. You can rectify this problem by defining the beam normals directly as explained previously.

Nodal offsets in beam sections

When beam elements are used as stiffeners for shell models, it is convenient to have the beam and shell elements share the same nodes. By default, shell element nodes are located at the midplane of the shell, and beam element nodes are located somewhere in the cross-section of the beam. Hence, if the shell and beam elements share the same nodes, the shell and the beam stiffener will overlap unless the beam cross-section is offset from the location of the node (see *Figure 1*).

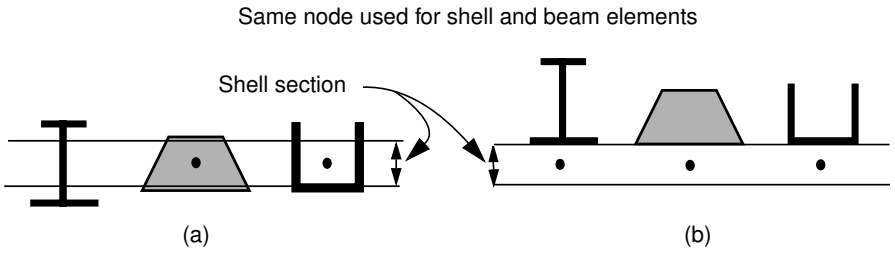


Figure 1: Using beams as stiffeners for shell models: (a) without offset of beam sections; (b) with offset of beam sections.

With beam section types I, TRAPEZOID, and ARBITRARY it is possible to specify that the section geometry is located at some distance from the origin of the section's local coordinate system, which is located at the element's nodes. Since it is easy to offset beams with such cross-sections from their nodes, they can be used readily as stiffeners as shown in *Figure 1*(b). (If flange or web buckling of the stiffeners is important, shells should be used to model the stiffeners.)

The I-beam shown in *Figure 2* is attached to a shell 1.2 units thick. The beam section can be oriented as it is shown in the figure by defining the offset of the beam node from the bottom of the I-section. The offset in this case would be −0.6; i.e., one half of the shell thickness.

You can also specify the location of the centroid and shear center; these locations can be offset from the beam node, thus enabling you to model stiffeners readily.

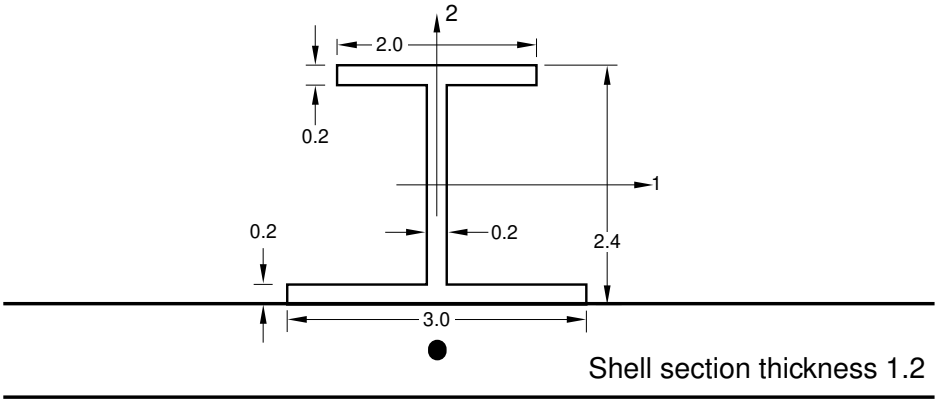


Figure 2: I-beam used as stiffener for a shell element.

It is also possible to define the beam nodes and shell nodes separately and connect the beam and shell using a rigid beam constraint between the two nodes. See *Linear Constraint Equations* for further details.

Formulation and integration

All beam elements in Abaqus are “beam-column” elements—meaning they allow axial, bending, and torsional deformation. The Timoshenko beam elements also consider the effects of transverse shear deformation.

In this section:

- *Shear deformation*
- *Torsional response—warping*

Shear deformation

The linear elements (B21 and B31) and the quadratic elements (B22 and B32) are shear deformable, Timoshenko beams; thus, they are suitable for modeling both stout members, in which shear deformation is important, and slender beams, in which shear deformation is not important. The cross-sections of these elements behave in the same manner as the cross-sections of the thick shell elements, as illustrated in *Figure 1*(b) and discussed in *Shell formulation – thick or thin*.

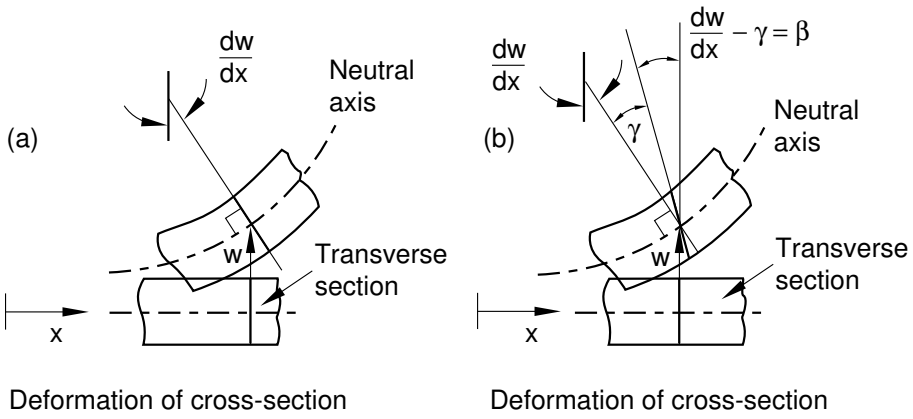


Figure 1: Behavior of transverse beam sections in (a) slender beams and (b) thick beams.

Abaqus assumes the transverse shear stiffness of these beam elements to be linear elastic and constant. In addition, these beams are formulated so that their cross-sectional area can change as a function of the axial deformation, an effect that is considered only in geometrically nonlinear simulations (see *Nonlinearity*) in which the section Poisson's ratio has a nonzero value. These elements can provide useful results as long as the cross-section dimensions are less than 1/10 of the typical axial dimensions of the structure, which is generally considered to be the limit of the applicability of beam theory; if the beam cross-section does not remain plane under bending deformation, beam theory is not adequate to model the deformation.

The cubic elements available in Abaqus/Standard—the so-called Euler-Bernoulli beam elements (B23 and B33)—do not model shear flexibility. The cross-sections of these elements remain perpendicular to the beam axis (see *Figure 1*(a)). Therefore, the cubic beam elements are most effective for modeling structures with relatively slender members. Since cubic elements model a cubic variation of displacement along their lengths, a structural member often can be modeled with a single cubic element for a static analysis and with a small number of elements for a dynamic analysis. These elements assume that shear deformations are negligible. Generally, if the cross-section dimensions are less than 1/15 of the typical axial dimensions of the structure, this assumption is valid.

Torsional response—warping

Structural members are often subjected to torsional moments, which occur in almost any three-dimensional frame structure. Loads that cause bending in one member may cause twisting in another, as shown in *Figure 1*.

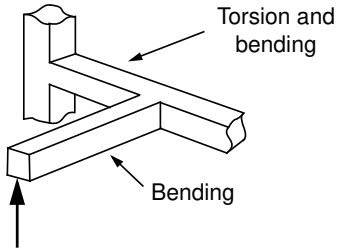


Figure 1: Torsion induced in a frame structure.

The response of a beam to torsion depends on the shape of its cross-section. Generally, torsion in a beam produces warping or nonuniform out-of-plane displacements in the cross-section. Abaqus considers the effects of torsion and warping only in the three-dimensional elements. The warping calculation assumes that the warping displacements are small. The following cross-sections behave differently under torsion: solid cross-sections; closed, thin-walled cross-sections; and open, thin-walled cross-sections.

Solid cross-sections

A solid, non-circular cross-section does not remain plane under torsion; instead, the section warps. Abaqus uses St. Venant warping theory to calculate a single component of shear strain caused by the warping at each section point in the cross-section. The warping in such solid cross-sections is considered unconstrained and creates negligible axial stresses. (Warping constraints would affect the solution only in the immediate vicinity of the constrained end.) The torsional stiffness of a beam with a solid cross-section depends on the shear modulus, G, of the material and the torsion constant, J, of the beam section. The torsion constant depends on the shape and the warping characteristics of the beam cross-section. Torsional loads that produce large amounts of inelastic deformation in the cross-section cannot be modeled accurately with this approach.

Closed, thin-walled cross-sections

Beams that have closed, thin-walled, non-circular cross-sections (BOX or HEX) have significant torsional stiffness and, thus, behave in a manner similar to solid sections. Abaqus assumes that warping in these sections is also unconstrained. The thin-walled nature of the cross-section allows Abaqus to consider the shear strains to be constant through the wall thickness. The thin-walled assumption is generally valid provided that the wall thickness is 1/10 a typical beam cross-section dimension. Examples of typical cross-section dimensions for thin-walled cross-sections include:

- The diameter of a pipe section.
- The length of an edge of a box section.
- The typical edge length of an arbitrary section.

Open, thin-walled cross-sections

Open, thin-walled cross-sections are very flexible in torsion when warping is unconstrained, and the primary source of torsional stiffness in such structures is the constraint of the axial warping strains. Constraining the warping of open, thin-walled beams introduces axial stresses that can affect the beam's response to other loading types. Abaqus/Standard has shear deformable beam elements, B31OS and B32OS, which include the warping effects in open, thin-walled sections. These elements must be used when modeling structures with open, thin-walled

cross-sections—such as a channel (defined as an ARBITRARY section) or an I-section—that are subjected to significant torsional loading.

The variation of the warping-induced axial deformation over the beam's cross-section is defined by the section's warping function. The magnitude of this function is treated as an extra degree of freedom, 7, in the open-section beam elements. Constraining this degree of freedom prevents warping at the nodes at which the constraints are applied.

So that the warping amplitude can be different in each branch, the junction between open-section beams in a frame structure generally should be modeled with separate nodes for each branch (see *Figure 2*).

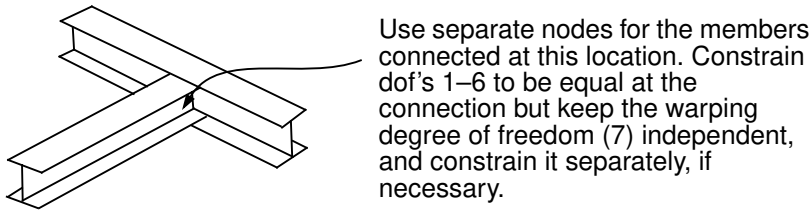


Figure 2: Connecting open-section beams.

However, if the connection is designed to prevent warping, all branches should share a common node, and the warping degree of freedom should be constrained.

A shear force that does not act through the beam's shear center produces torsion. The twisting moment is equal to the shear force multiplied by its eccentricity with respect to the shear center. Often, the centroid and the shear center do not coincide in open, thin-walled beam sections (see *Figure 3*). If the nodes are not located at the shear center of the cross-section, the section may twist under loading.

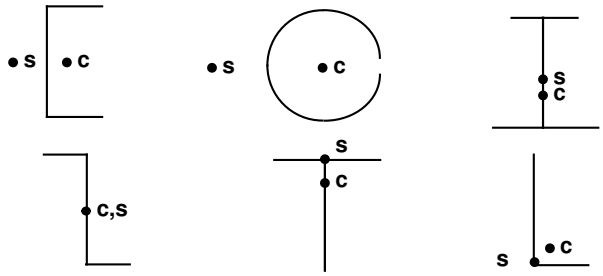


Figure 3: Approximate locations of shear centers, *s*, and centroids, *c*, for a number of beam cross-sections.

Selecting beam elements

Several guidelines are provided for selecting beam elements.

- First-order, shear-deformable (Timoshenko) beam elements (B21, B31) should be used in any simulation that includes contact.
- If the transverse shear deformation is important, use Timoshenko beam elements (B21, B22, B31, B32).
- If the structure is either very rigid or very flexible, the hybrid beam elements (B21H, B32H, etc.) available in Abaqus/Standard should be used in geometrically nonlinear simulations.
- The Euler-Bernoulli (cubic) beams (B23, B33) available in Abaqus/Standard are very accurate for simulations that include distributed loading, such as dynamic vibration analyses.
- Structures with open, thin-walled cross-sections should be modeled with the elements that use open-section warping theory (B31OS, B32OS) available in Abaqus/Standard.

Example: cargo crane

This example models a light-service, cargo crane.

In this section:

- *Cargo crane analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*

Cargo crane analysis

You have been asked to determine the static deflections of the crane when it carries a load of 10 kN.
You should also identify the critical members and joints in the structure; i.e., those with the highest stresses and loads.
Because this is a static analysis you will analyze the cargo crane using Abaqus/Standard.

A light-service cargo crane is shown in *Figure 1*. The crane consists of two truss structures joined together by cross bracing. The two main members in each truss structure are steel box beams (box cross-sections). Each truss structure is stiffened by internal bracing, which is welded to the main members. The cross bracing connecting the two truss structures is bolted to the truss structures. These connections can transmit little, if any, moment and, therefore, are treated as pinned joints.

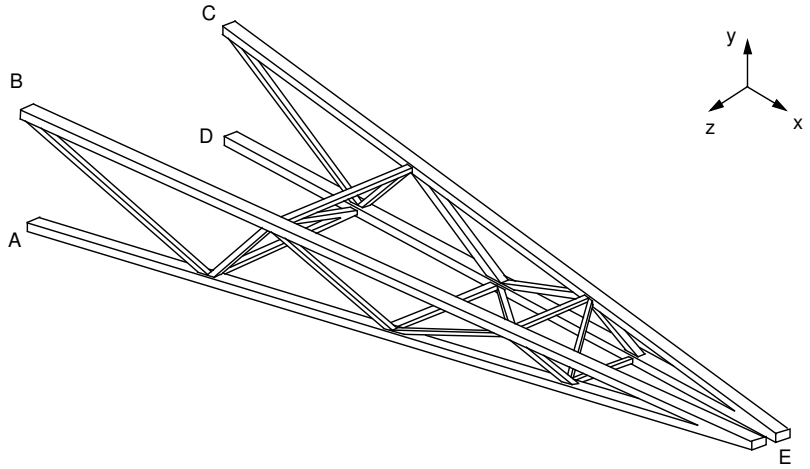


Figure 1: Sketch of a light-service cargo crane.

Both the internal bracing and cross bracing use steel box beams with smaller cross-sections than the main members of the truss structures. The two truss structures are connected at their ends (at point E) in such a way that allows independent movement in the 3-direction and all of the rotations, while constraining the displacements in the 1- and 2-directions to be the same. The crane is welded firmly to a massive structure at points A, B, C, and D. The dimensions of the crane are shown in *Figure 2*. In the following figures, truss A is the structure consisting of members AE, BE, and their internal bracing; and truss B consists of members CE, DE, and their internal bracing.

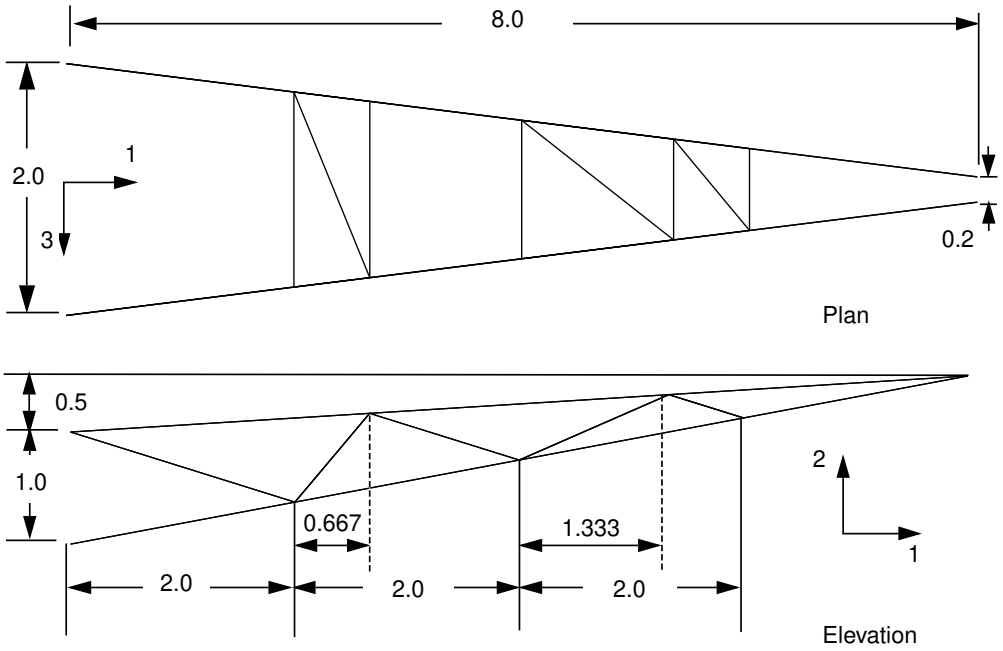


Figure 2: Dimensions (in m) of the cargo crane.

The ratio of the typical cross-section dimension to global axial length in the main members of the crane is much less than 1/15. The ratio is approximately 1/15 in the shortest member used for internal bracing. Therefore, it is valid to use beam elements to model the crane.

Preprocessing—creating the model with Abaqus/CAE

In this section we discuss how to use Abaqus/CAE to create the entire model for this simulation. Abaqus provides scripts that replicate the complete analysis model for this problem.
Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Cargo crane*. Instructions of how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**, highlight **Cargo crane**, and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

In this section:

- *Creating the parts*
- *Defining beam section properties*
- *Defining beam section orientations*
- *Beam normals*
- *Creating assembly-level sets*
- *Creating a step definition and specifying output*
- *Defining constraint equations*
- *Modeling the pin joint between the cross bracing and the trusses*
- *Defining loads and boundary conditions*
- *Creating the mesh*
- *Using the Keywords Editor and defining the job*

Creating the parts

The welded joints between the internal bracing and main members in the crane provide complete continuity of the translations and rotations from one region of the model to the next. Therefore, you need only a single geometric entity (i.e., vertex) at each welded joint in the model. A single part is used to represent the internal bracing and main members. For convenience, both truss structures will be treated as a single part.

The bolted joints, which connect the cross bracing to the truss structures, and the connection at the tip of the truss structures are different from the welded joint connections. Since these joints do not provide complete continuity for all degrees of freedom, separate vertices are needed for connection. Thus, the cross bracing must be treated as a separate part since distinct geometric entities are required to model the bolted joints. Appropriate constraints between the separate vertices must be specified.

We begin by discussing a technique to define the truss geometry. Since the two truss structures are identical, it is sufficient to define the base feature of the part using only the geometry of a single truss structure. The sketch of the truss geometry can be saved and then used to add the second truss structure to the part definition.

The dimensions shown in *Figure 2* are relative to a global Cartesian coordinate system. The base feature, however, must be sketched in a local plane. To make the sketching easier, datum features will be used. A datum plane, parallel to one of the trusses (truss B in *Figure 1*, for example), will serve as the sketch plane. The orientation of the sketch plane will be defined using a datum axis.

The other truss will also be added as a planar wire feature by projecting the truss created here onto a new datum plane.

The final truss part is shown in *Figure 1*. The visibility of all datum and reference geometry has been suppressed.

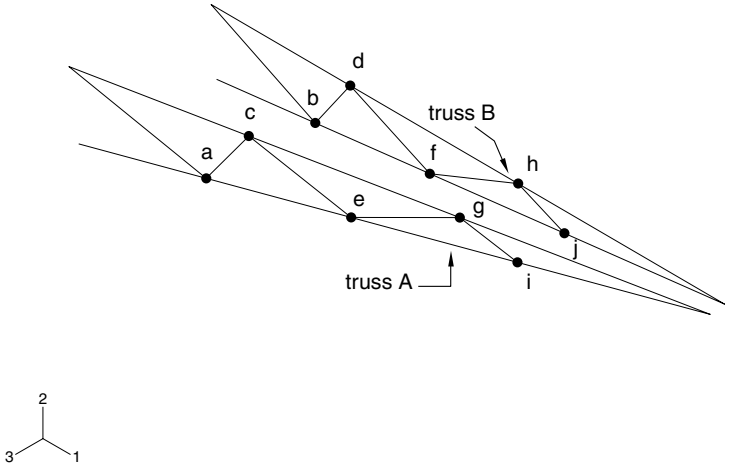


Figure 1: Final geometry of the truss structures; highlighted vertices indicate the locations of the pin joints.

Define the geometry of a single truss

1. To create a datum plane, a part must first be created. A part consisting of a single reference point will serve this purpose. Begin by creating a three-dimensional deformable part using the point base feature. Set the approximate part size to `20.0`, and name the part `Truss`. Place the point at the origin. This point represents point D in *Figure 1*.

2. Using the **Create Datum Point: Offset From Point** tool , create two datum points at distances of `(0, 1, 0)` and `(8, 1.5, 0.9)` from the reference point. These points represent points C and

E, respectively, in *Figure 1*. Reset the view using the **Auto-Fit View** tool in the View Manipulation toolbar to see the full model.

3. Using the **Create Datum Plane: 3 Points** tool , create a datum plane to serve as the sketch plane. Select the reference point first, and then select the other two datum points in a counterclockwise fashion. Click mouse button 2 to exit the procedure.

   Note: While selecting the points in this way is not required, it will make certain operations that follow easier. For example, by selecting the points in a counterclockwise order, the normal to the plane points out of the viewport and the sketch plane will be oriented automatically in the 1–2 view in the Sketcher. If you select the points in a clockwise order, the plane's normal will point into the viewport and the sketch plane will have to be adjusted in the Sketcher.

4. Using the **Create Datum Axis: Principal Axis** tool , create a datum axis parallel to the **Y-Axis**. As noted earlier, this axis will be used to position the sketch plane.

5. You are now ready to sketch the geometry. Use the **Create Wire: Planar** tool to enter the Sketcher. Select the datum plane as the plane on which to sketch the wire geometry; select the datum axis as the axis that will appear vertical and to the left of the sketch. You may need to resize the view to select these entities.

6. Once in the Sketcher, use the **Sketcher Options** tool to modify the display. In the **General** tab, change the **Sheet size** to `20` and reduce the **Grid spacing** to `2`. Zoom in to see the datum points more clearly.

   Note: If the sketch plane is not oriented in the 1–2 plane, use the Views toolbar to change to the X–Y view.

   Using the **Create Lines: Connected** tool , sketch the lines representing the main truss, as shown in *Figure 2*. The datum points that were projected are treated as fixed points in the sketch. Any line connected to one of these points effectively inherits a fixed constraint at that point.

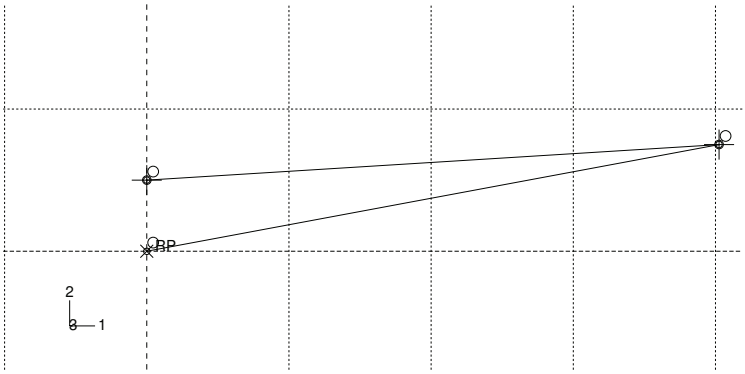


Figure 2: Main members of the truss.

7. Next, create a series of connected lines as shown in *Figure 3* to approximate the interior bracing of the truss.

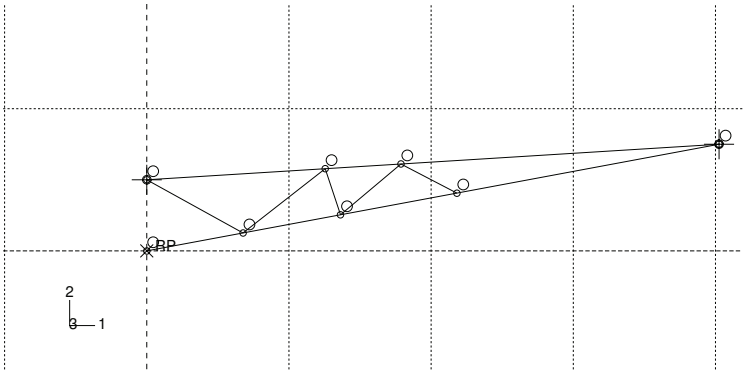


Figure 3: Rough layout of interior members.

At this stage, the layout of the interior bracing is arbitrary and is intended only as a rough approximation of the true shape. The endpoints of the lines, however, must snap to the edges of the main truss members. This is indicated in the figure by the presence of small circles next to the intersections of the interior bracing with the main members. Avoid creating 90° angles because that will introduce unwanted additional constraints.

8. Split the edges of the main members at the points where they intersect the interior bracing.
9. Dimension the vertical distance between the left endpoints of the sketch and the horizontal distance between the reference point and the right endpoint of the sketch, as shown in *Figure 4*. These dimensions will act as additional constraints on the sketch. Accept the values shown in the prompt area when creating the dimensions. These values represent the dimensions of the part, projected from the global Cartesian coordinate system (depicted in *Figure 2*) to the local sketch plane.

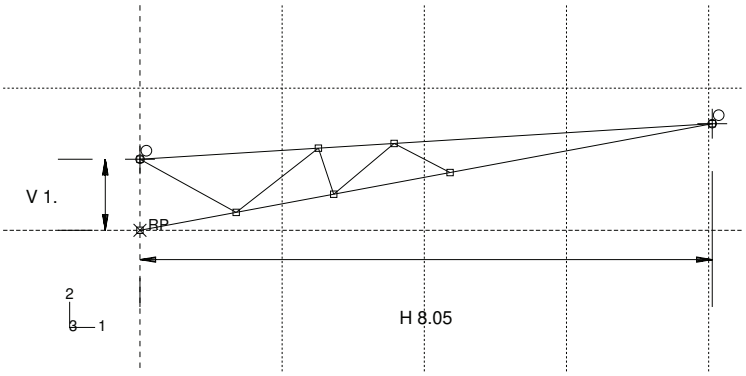


Figure 4: Dimensioned sketch.

10. Apply parallel constraints to the segments of the top edge of the main member, then repeat this operation for the bottom edge of the main member. These constraints ensure that these line segments remain colinear.
11. To complete the sketch, recognize from *Figure 2* that the interior bracing breaks the main members into equal length segments on both its top and bottom edges. Thus, impose equal length constraints on the segments of the top edge of the main member; repeat this operation for the bottom edge of the main member. The final sketch appears as shown in *Figure 5*.

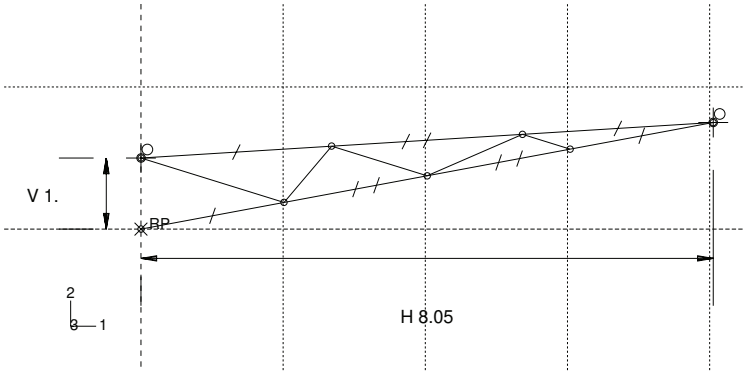


Figure 5: Final sketch of single truss structure.

12. Using the **Save Sketch As** tool , save the sketch as `Truss`.
13. Click **Done** to exit the Sketcher and to save the base feature of the part.

Define the geometry of the second truss structure

1. Define three datum points using offsets from the end points of the truss, as shown in *Figure 6*. The offsets from the parent vertices are indicated in the figure. You may need to rotate your sketch to see the datum points.

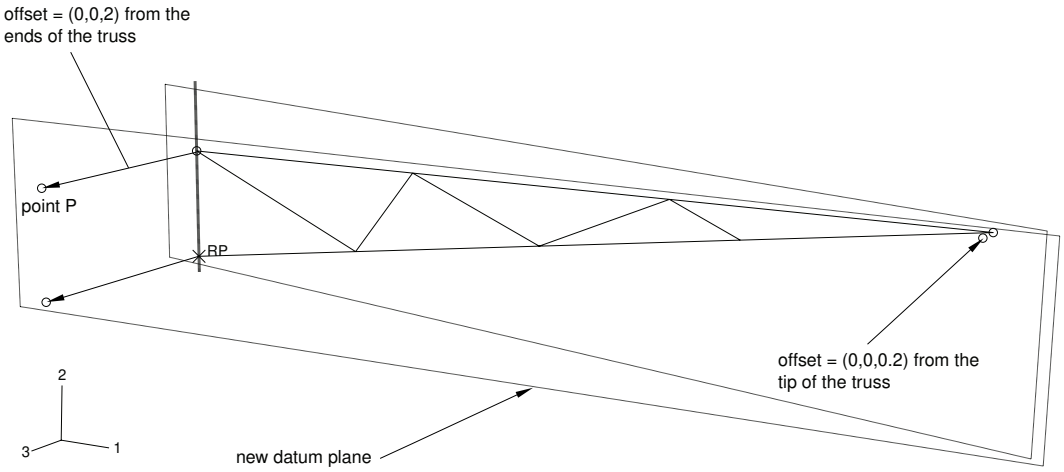


Figure 6: Datum points, plane, and axis.

2. Create a datum plane using these three points. As before, the points defining the plane should be chosen in a counterclockwise order.
3. Use the **Create Wire: Planar** tool to add a feature to the part. Select the new datum plane as the sketch plane and the datum axis created earlier as the edge that will appear vertical and on the left of the sketch.



Note:

If the sketch plane is not oriented in the 1–2 plane, use the Views toolbar to change to the X–Y view.

4. Use the **Add Sketch** tool to retrieve the truss sketch. Translate the sketch by selecting the vertex at the top left end of the new truss as the starting point of the translation vector and the datum point labeled `P` in *Figure 6* as the endpoint of the vector. Zoom in and rotate the view as necessary to facilitate your selections.

   Note: If the points defining either the original or new datum plane were not selected in a counterclockwise order, you will have to mirror the sketch before translating it. If necessary, cancel the sketch retrieval operation, create the necessary construction line for mirroring, and retrieve the sketch again.

5. Click **Done** in the prompt area to exit the Sketcher.

Create the cross brace geometry

Recall that the cross bracing must be treated as a separate part to properly represent the pin joints between it and the trusses. The easiest way to sketch the cross bracing, however, is to create wire features directly between the locations of the joints in the trusses. Thus, we will adopt the following method to create the cross bracing part: first, a copy of the truss part will be created and the wires representing the cross brace will be added to it (we cannot use this new part as is because the vertices at the joints are shared and, thus, cannot represent a pin joint); then, we will use the cut feature available in the Assembly module to perform a Boolean cut between the truss with the cross brace and the truss without the cross brace, leaving the cross brace geometry as a distinct part. The procedure is described in detail below.

1. In the Model Tree, click mouse button 3 on the **Truss** item underneath the **Parts** container and select **Copy** from the menu that appears. In the **Part Copy** dialog box, name the new part `Truss-all`, and click **OK**.

2. The pin locations are highlighted in *Figure 1*. Use the **Create Wire: Point to Point** tool . In the **Create Wire Feature** dialog box, accept the default setting of **Chained wires** and click to add the cross bracing geometry to the new part, as shown in *Figure 7* (the vertices in this figure correspond to those labeled in *Figure 1*; the visibility of the truss in *Figure 7* has been suppressed). Use the following coordinates to specify a similar view: **Viewpoint** (1.19, 5.18, 7.89), **Up vector** (–0.40, 0.76, –0.51).

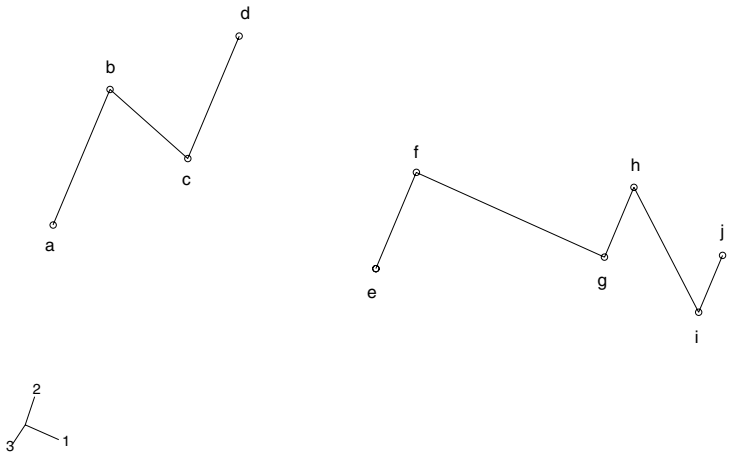


Figure 7: Cross bracing geometry.

Tip: If you make a mistake while connecting the cross bracing geometry, you can delete a line using the **Delete Feature** tool ; you cannot recover deleted features.

3. Create an instance of each part (`Truss` and `Truss-all`).
4. From the main menu bar of the Assembly module, select **Instance->Merge/Cut**. In the **Merge/Cut Instances** dialog box, name the new part `Cross brace`, select **Cut geometry** in the **Operations** field, and click **Continue**.
5. In the prompt area, click **Instances**. Select `Truss-all-1` as the instance to be cut. Similarly, select `Truss-1` as the instance that will make the cut.

   After the cut is made, a new part named `Cross brace` is created that contains only the cross brace geometry. The current model assembly contains only an instance of this part; the original part instances are suppressed by default. Since we will need to use the original truss in the model assembly, click mouse button 3 on **Truss-1** underneath the **Instances** container and select **Resume** from the menu that appears to resume this part instance.

   We now define the beam section properties.

Defining beam section properties

Since the material behavior in this simulation is assumed to be linear elastic, it is more efficient from a computational point of view to precompute the beam section properties. Assume the trusses and bracing are made of a mild strength steel with $E = 200.0 \times 10^9$ Pa, $\nu = 0.25$, and $G = 80.0 \times 10^9$ Pa. All the beams in this structure have a box-shaped cross-section.

A box-section is shown in *Figure 1*. The dimensions shown in *Figure 1* are for the main members of the two trusses in the crane. The dimensions of the beam sections for the bracing members are shown in *Figure 2*.

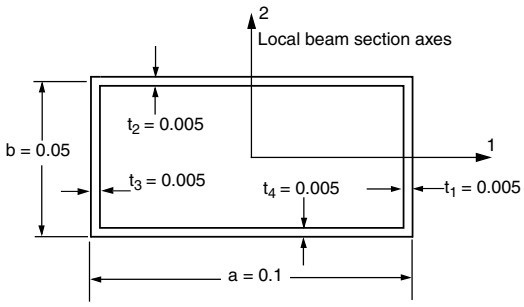


Figure 1: Cross-section geometry and dimensions (in m) of the main members.

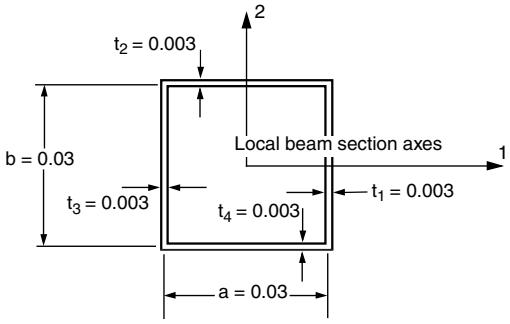


Figure 2: Cross-section geometry and dimensions (in m) of the internal and cross bracing members.

1. In the Model Tree, double-click the **Profiles** container to create a box profile for the main members of the truss structures; then, create a second profile for the internal and cross bracing. Name the profiles `MainBoxProfile` and `BraceBoxProfile`, respectively. Use the dimensions shown in *Figure 1* and *Figure 2* to complete the profile definitions.
2. Create one **Beam** section for the main members of the truss structures and one for the internal and cross bracing. Name the sections `MainMemberSection` and `BracingSection`, respectively.
   a. For both section definitions, specify that section integration will be performed before the analysis. When this type of section integration is chosen, material properties are defined as part of the section definition rather than in a separate material definition.
   b. Choose `MainBoxProfile` for the main members' section definition, and `BraceBoxProfile` for the bracing section definition.
   c. Click the **Basic** tab, and enter the Young's and shear moduli noted earlier in the appropriate fields of the data table.
   d. Enter the **Section Poisson's ratio** in the appropriate text field of the **Edit Beam Section** dialog box.

3. Assign `MainMemberSection` to the geometry regions representing the main members of the trusses and `BracingSection` to the regions representing the internal and cross bracing members. Use the **Part** list located in the context bar to retrieve each part. You can ignore the `Truss-all` part since it is no longer needed.

Defining beam section orientations

The beam section axes for the main members should be oriented such that the beam 1-axis is orthogonal to the plane of the truss structures shown in the elevation view (*Figure 2*) and the beam 2-axis is orthogonal to the elements in that plane. The approximate $\mathbf{n_1}$-vector for the internal truss bracing is the same as for the main members of the respective truss structures.

In its local coordinate system, the `Truss` part is oriented as shown in *Figure 1*.

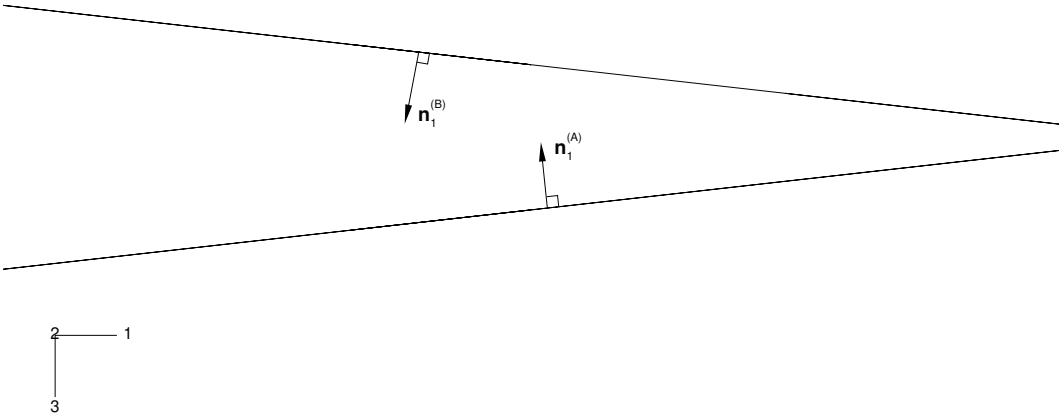


Figure 1: Orientation of the truss in its local coordinate system.

From the main menu bar of the Property module, select **Assign->Beam Section Orientation** to specify an approximate $\mathbf{n_1}$-vector for each truss structure. As noted earlier, the direction of this vector should be orthogonal to the plane of the truss. Thus, for truss B, the approximate $\mathbf{n_1}$ = (–0.1118, 0.0, 0.9936); while for the other truss structure (truss A), the approximate $\mathbf{n_1}$ = (–0.1118, 0.0, –0.9936).

You may want to check that your beam sections and orientations are correct. From the main menu bar, select **View->Part Display Options** and toggle on **Render beam profiles** to see a graphical representation of the beam profile. Toggle off **Render beam profiles** before continuing with the rest of the example. This functionality is also available in the Visualization module through the **ODB Display Options** dialog box.

From the main menu bar, select **Assign->Element Tangent** to specify the beam tangent directions. Flip the tangent directions as necessary so that they appear as shown in *Figure 2*.

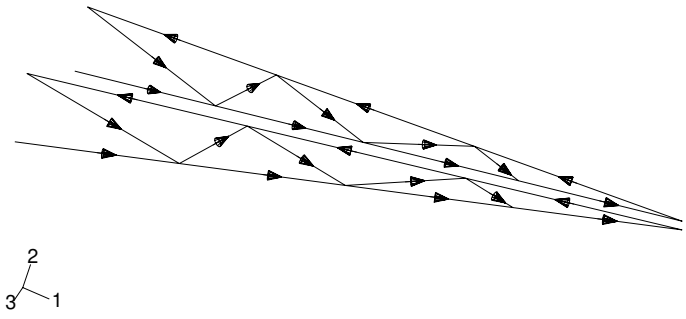


Figure 2: Beam tangent directions.

While both the cross bracing and the bracing within each truss structure have the same beam section geometry, they do not share the same orientation of the beam section axes. Since the square cross bracing members are subjected to

primarily axial loading, their deformation is not sensitive to cross-section orientation; thus, we make some assumptions so that the orientation of the cross-bracing is somewhat easier to specify. All of the beam normals ($\mathbf{n_2}$-vectors) should lie approximately in the plane of the plan view of the cargo crane (see *Figure 2*). This plane is skewed slightly from the global 1–3 plane. A simple method for defining such an orientation is to provide an approximate $\mathbf{n_1}$-vector that is orthogonal to this plane. The vector should be nearly parallel to the global 2-direction. Therefore, specify $\mathbf{n_1}$ = (0.0, 1.0, 0.0) for the cross bracing so that it is aligned with the part (and as we shall see later, global) *y*-axis.

Beam normals

In this model you will have a modeling error if you provide data that only define the orientation of the approximate $\mathbf{n_1}$-vector. Unless overridden, the averaging of beam normals (see *Beam element curvature*) causes Abaqus to use incorrect geometry for the cargo crane model. To see this, you can use the Visualization module to display the beam section axes and beam tangent vectors (see *Postprocessing*). Without any further modification to the beam normal directions, the normals in the crane model would appear to be correct in the Visualization module; yet, they would be, in fact, slightly incorrect.

Figure 1 shows the geometry of the truss structure.

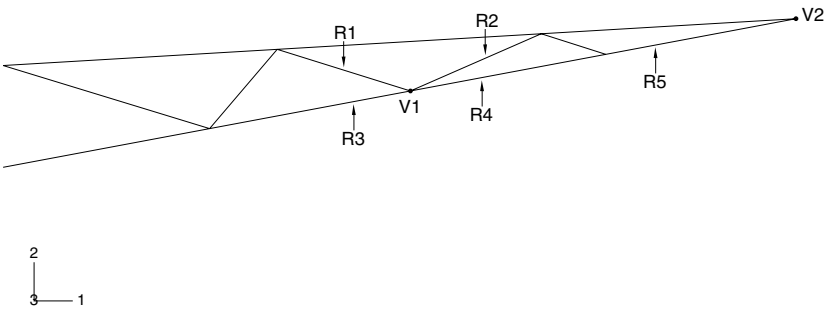


Figure 1: Locations where beam normals should be specified.

Referring to this figure, the correct geometry for the crane model requires three independent beam normals at vertex V1: one each for regions R1 and R2 and a single normal for regions R3 and R4. Using the Abaqus logic for averaging normals, it becomes readily apparent that the beam normal at vertex V1 in region R2 would be averaged with the normals at this point for the adjacent regions. In this case the important part of the averaging logic is that normals that subtend an angle less than 20° with the reference normal are averaged with the reference normal to define a new reference normal. Assume the original reference normal at this point is the normal for regions R3 and R4. Since the normal at vertex V1 in region R2 subtends an angle less than 20° with the original reference normal, it is averaged with the original normal to define a new reference normal at that location. On the other hand, since the normal at vertex V1 in region R1 subtends an angle of approximately 30° with the original reference normal, it will have an independent normal.

As a result of the incorrect average normal, the elements created on regions R2, R3, and R4 that share the node created at vertex V1 would have an intended geometry that exhibits curvature from one end of the element to the other. To prevent Abaqus from applying its averaging algorithm, specify the normal directions explicitly at positions where the subtended angle between adjacent regions will be less than 20°. In this problem you must specify the normal positions for the corresponding regions on both sides of the crane.

There is also a problem with the normals at vertex V2 at the tip of the truss structure, again because the angle between the two regions attached to this vertex is less than 20°. Since we are modeling straight beams, the normals are constant at both ends of each beam. This can be corrected by explicitly specifying the beam normal direction. As before, you must do this for the corresponding regions on both sides of the crane.

Currently, the only way to specify beam normal directions in Abaqus/CAE is with the **Keywords Editor**. The **Keywords Editor** is a specialized text editor that allows you to modify the Abaqus input file generated by Abaqus/CAE before submitting it for analysis. Thus, it allows you to add Abaqus/Standard or Abaqus/Explicit functionality when such functionality is not supported by the current release of Abaqus/CAE. For more information on the **Keywords Editor**, see *Adding unsupported keywords to your Abaqus/CAE model*.

You will specify the beam normal directions later.

Creating assembly-level sets

We now focus on assembling the model.

Since the parts are already aligned with the global Cartesian coordinate system shown in *Figure 1*, no further manipulations of the parts are necessary.

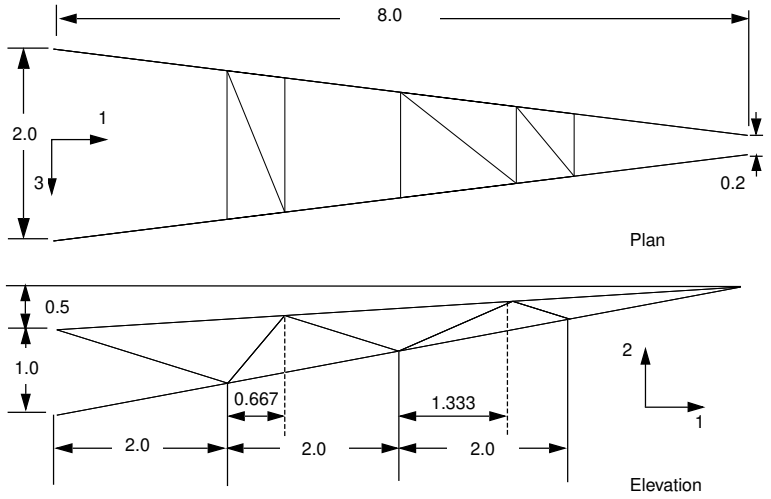


Figure 1: Dimensions (in m) of the cargo crane.

At this point, however, it is convenient to define assembly-level geometry sets that will be used later. In the Model Tree, expand the **Assembly** container and double-click **Sets**. Define a geometry set containing the vertices corresponding to points A through D (refer to *Figure 2* for the exact locations), and name the set `Attach`. When defining this set, be sure to select the vertices of the truss and not reference points. You may need to use the **Selection** toolbar to aid in your selection.

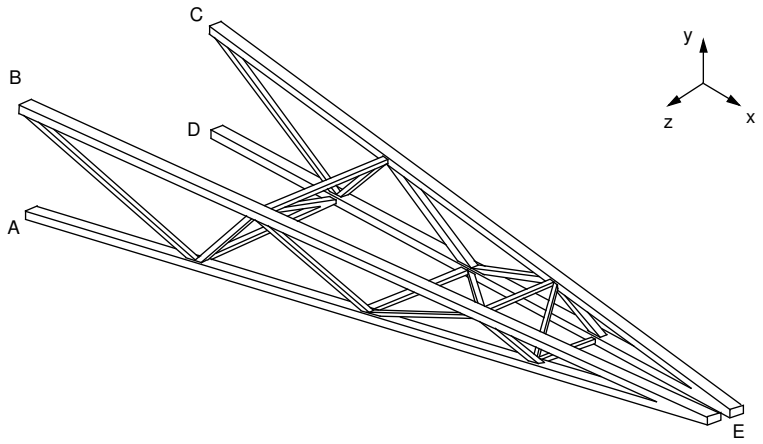
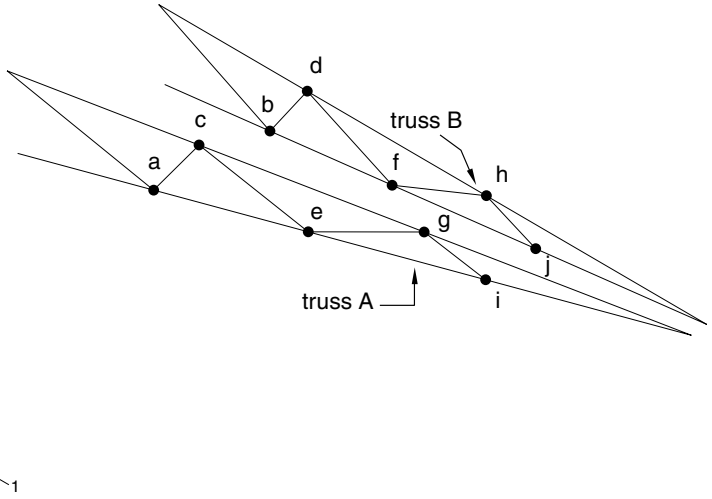


Figure 2: Sketch of a light-service cargo crane.

In addition, create sets at the vertices located at the tips of the trusses (location E in *Figure 2*). Name the sets `Tip-a` and `Tip-b`, with `Tip-a` being the geometry set associated with truss A (see *Figure 3*). Finally, create a set for each region where beam normals will be specified, referring to *Figure 3*. For truss A, create a set named `Inner-a` for the

region indicated by R2 and a set named `Leg-a` for the region indicated by R5; create corresponding sets `Inner-b` and `Leg-b` for truss B.

Figure 3: Final geometry of the truss structures; highlighted vertices indicate the locations of the pin joints.



Creating a step definition and specifying output

You can now define the analysis step.

Create a single static, general step. Name the step `Tip load`, and enter the following step description: `Static tip load on crane`.

Write the displacements (U) and reaction forces and moments (RF) at the nodes and the section forces and moments (SF) in the elements to the output database as field output for postprocessing with Abaqus/CAE.

Defining constraint equations

Constraints between nodal degrees of freedom are specified in the Interaction module. The form of each equation is

$$A_1 u_1 + A_2 u_2 + \ldots + A_n u_n = 0,$$

where A_i is the coefficient associated with degree of freedom u_i.

In the crane model the tips of the two trusses are connected together such that degrees of freedom 1 and 2 (the translations in the 1- and 2-directions) of each tip node are equal, while the other degrees of freedom (3–6) are independent. We need two linear constraints, one equating degree of freedom 1 at the two vertices and the other equating degree of freedom 2.

The degrees of freedom associated with the first set defined in an equation are eliminated from the stiffness matrix. Therefore, this set should not appear in other constraint equations, and boundary conditions should not be applied to the eliminated degrees of freedom.

1. In the Model Tree, double-click the **Constraints** container. Name the constraint `TipConstraint-1`, and specify an equation constraint.
2. In the **Edit Constraint** dialog box, enter a coefficient of `1.0`, the set name `Tip-a`, and degree of freedom `1` in the first row. In the second row, enter a coefficient of `-1.0`, the set name `Tip-b`, and degree of freedom `1`. Click **OK**.

   This defines the constraint equation for degree of freedom 1.

   Note:

   Text input is case-sensitive in Abaqus/CAE.

3. Click mouse button 3 on the **TipConstraint-1** item underneath the **Constraints** container, and select **Copy** from the menu that appears. Copy `TipConstraint-1` to `TipConstraint-2`.
4. Double-click **TipConstraint-2** underneath the **Constraints** container to edit it. Change the degree of freedom on both lines to `2`.

Modeling the pin joint between the cross bracing and the trusses

The cross bracing, unlike the internal truss bracing, is bolted to the truss members. You can assume that these bolted connections are unable to transmit rotations or torsion. The duplicate vertices that were defined at these locations are needed to define this constraint. In Abaqus such constraints can be defined using multi-point constraints, constraint equations, or connectors. In this example the last approach is adopted.

Connectors allow you to model a connection between any two points in a model assembly (or between any single point in the assembly and the ground). A large library of connectors is available in Abaqus. See *Connector Element Library* for a complete list and a description of each connector type.

The JOIN connector will be used to model the bolted connection. The pinned joint created by this connector constrains the displacements to be equal but the rotations (if they exist) remain independent.

In Abaqus/CAE connectors are modeled using connector section assignments. You create an assembly-level wire feature to define the connector geometry and a connector section to define the connection type. You can model connectors at all of the pin joints using one connector section assignment. You create the connector section assignment by selecting multiple wires and specifying a connector section to assign to the selected wires (similar to the association between elements and their section properties). For connections between coincident points, a specialized tool is available to simplify the modeling process. This tool is used here.

1. From the main menu bar in the Interaction module, select **Connector->Coincident Builder**.
2. Use the **Selection** toolbar to restrict your selection to **Vertices**.
3. In the viewport, drag the cursor around the entire model, then click **Done** in the prompt area.

   Ten pairs of points should appear in the **Coincident Point Builder** table. Each corresponds to one of the pin joint locations labeled `a–j` in *Figure 1*.

4. In the **Coincident Point Builder** dialog box, click  to create a section property.
5. In the **Create Connector Section** dialog box, select **Basic** as the **Connection Category**. From the list of available translational types, select **Join**. Accept all other defaults, and click **Continue**.
6. No additional section specifications are required. Thus, in the connector section editor that appears, click **OK**.
7. Click **OK** in the **Coincident Point Builder** dialog box to create the connections.

Defining loads and boundary conditions

A total load of 10 kN is applied in the negative *y*-direction to the ends of the truss.

Recall there is a constraint equation connecting the *y*-displacement of sets `Tip-a` and `Tip-b`, where the degree of freedom for set `Tip-a` is eliminated from the system equations. Thus, apply the load as a concentrated force of magnitude `-10000` to set `Tip-b`. Name the load `Tip load`. Because of the constraint, the load will be carried equally by both trusses.

The crane is attached firmly to the parent structure. Create an encastre boundary condition named `Fixed end`, and apply it to set `Attach`.

Creating the mesh

The cargo crane will be modeled with three-dimensional, slender, cubic beam elements (B33). The cubic interpolation in these elements allows us to use a single element for each member and still obtain accurate results under the applied bending load. The mesh that you should use in this simulation is shown in *Figure 1*.

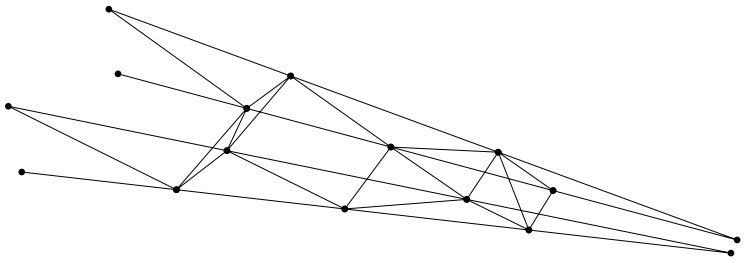


Figure 1: Mesh for cargo crane.

In the Model Tree, expand the **Truss** item underneath the **Parts** container. Then double-click **Mesh** in the list that appears. Specify a global part seed of `2.0` for all regions. Repeat this for the part named **Cross brace**.

Tip: In the context bar of the Mesh module, select the appropriate part from the **Object** pull-down list to switch between the parts more readily.

Mesh both part instances with linear cubic beams in space (B33 elements).

Using the Keywords Editor and defining the job

You will now add the keyword options that are necessary to complete the model definition (namely, the option to define beam normal directions) using the **Keywords Editor**. Refer to the *Abaqus Keywords Guide* for a description of the required syntax, if necessary.

1. In the Model Tree, click mouse button 3 on **Model-1** and select **Edit Keywords** from the menu that appears.

   The **Edit Keywords** dialog box appears containing the input file that has been generated for your model.

2. In the **Keywords Editor** each keyword is displayed in its own block. Only text blocks with a white background can be edited. Select the text block that appears just prior to the **END ASSEMBLY* option. Click **Add After** to add an empty block of text.

3. In the block of text that appears, enter the following:

   ```
   *NORMAL, TYPE=ELEMENT
   Inner-a,  Inner-a, -0.3962,  0.9171,  0.0446
   Inner-b,  Inner-b,  0.3962, -0.9171,  0.0446
   Leg-a,    Leg-a,   -0.1820,  0.9829,  0.0205
   Leg-b,    Leg-b,    0.1820, -0.9829,  0.0205
   ```

   Tip: Copy and paste data from one location in a text block to another.

4. When you have finished, click **OK** to exit the **Keywords Editor**.

   Before continuing, rename the model to `Static`. This model will later form the basis of the model used in the crane example discussed in *Linear Dynamics*.

   Save your model in a model database file named `Crane.cae`, and create a job named `Crane`.

   Submit the job for analysis, and monitor the solution progress. If any modeling errors are encountered, correct them; investigate the cause of any warning messages, taking appropriate action as necessary.

Postprocessing

Switch to the Visualization module, and open the file `Crane.odb`. Abaqus displays an undeformed shape plot of the crane model.

Plotting the deformed model shape

To begin this exercise, plot the deformed model shape with the undeformed model shape superimposed on it. Specify a nondefault view using (0, 0, 1) as the *X*-, *Y*-, and *Z*-coordinates of the viewpoint vector and (0, 1, 0) as the *X*-, *Y*-, and *Z*-coordinates of the up vector.



Tip: You can also display the model using this view by clicking  from the **Views** toolbar.

The undeformed shape of the crane superimposed upon the deformed shape is shown in *Figure 1*.

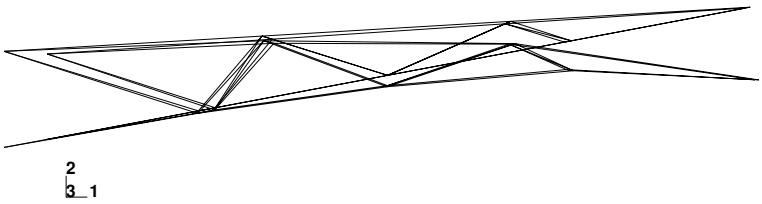


Figure 1: Deformed shape of cargo crane.

Using display groups to plot element and node sets

You can use display groups to plot existing node and element sets; you can also create display groups by selecting nodes or elements directly from the viewport. You will create a display group containing only the elements associated with the main members in truss structure A.

To create and plot a display group:

1. In the Results Tree, expand the **Sections** container underneath the output database file named `Crane.odb`.
2. To facilitate your selection, change the view back to the default isometric view using the  tool in the Views toolbar.

   Tip: If the Views toolbar is not visible, select **View->Toolbars->Views** from the main menu bar.
3. In succession, click the items in the container until the elements associated with the main members in truss A are highlighted in the viewport. Click mouse button 3 on this item and select **Replace** from the menu that appears.

   Abaqus/CAE now displays only this group of elements.
4. To save this group, double-click **Display Groups** in the Results Tree; or use the  tool in the Display Group toolbar.

   The **Create Display Group** dialog box appears.

5. In the **Create Display Group** dialog box, click **Save As** and enter `MainA` as the name for your display group.
6. Click **Dismiss** to close the **Create Display Group** dialog box.

   This display group now appears underneath the **Display Groups** container in the Results Tree.

Beam cross-section orientation

You will now plot the section axes and beam tangents on the undeformed model shape.

To plot the beam section axes:

1. From the main menu bar, select **Plot->Undeformed Shape**; or use the  tool in the toolbox to display only the undeformed model shape.
2. From the main menu bar, select **Options->Common**; then, click the **Normals** tab in the dialog box that appears.
3. Toggle on **Show normals**, and accept the default setting of **On elements**.
4. In the **Style** area at the bottom of the **Normals** page, specify the **Length** to be **Long**.
5. Click **OK**.

   The section axes and beam tangents are displayed on the undeformed shape.

The resulting plot is shown in *Figure 2*. The text annotations in *Figure 2* that identify the section axes and beam tangent will not appear in your image. The vector showing the local beam 1-axis, $\mathbf{n_1}$, is blue; the vector showing the beam 2-axis, $\mathbf{n_2}$, is red; and the vector showing the beam tangent, $\mathbf{t}$, is white.

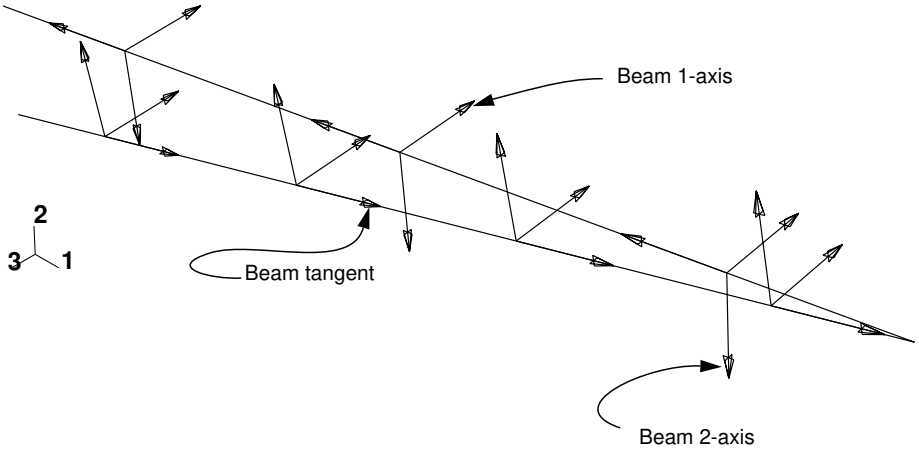


Figure 2: Plot of beam section axes and tangents for elements in display group `MainA`.

Rendering beam profiles

You will now display an idealized representation of the beam profile and contour the stress results.

To render beam profiles:

1. From the main menu bar, select **View->ODB Display Options**.

   The **ODB Display Options** dialog box appears.

2. In the **General** tabbed page, toggle on **Render beam profiles** and accept the default scale factor of 1.
3. Click **OK**.

   Abaqus/CAE displays beam profiles with the appropriate dimensions and in the correct orientations. *Figure 3* shows the beam profiles on the whole model. Your changes are saved for the duration of the session.

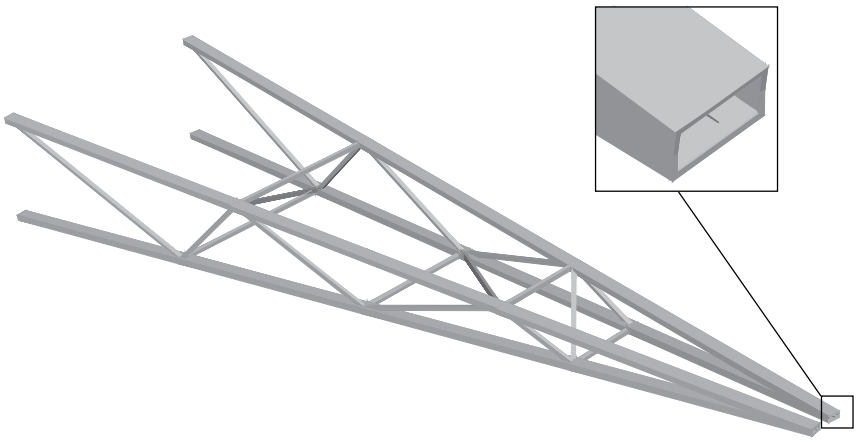


Figure 3: Cargo crane with beam profiles displayed.

Creating a hard copy

You can save the current image to a file for hard-copy output.

To create a PNG file of the current image:

1. From the main menu bar, select **File->Print**.

   The **Print** dialog box appears.
2. From the **Settings** area in the **Print** dialog box, select **Color** as the **Rendition** type; and toggle on **File** as the **Destination**.
3. Select **PNG** as the **Format**, and enter `beam` as the **File name**.
4. In the **Print** dialog box, click **OK**.

   Abaqus/CAE creates a PNG file of the current image and saves it in your working directory as `beam.png`. You can print this file using your system's command for printing PNG files.

Displacement summary

Write a summary of the displacements of all nodes in display group `MainA` to a file named `crane.rpt`. The peak displacement at the tip of the crane in the 2-direction is 0.0188 m.

Section forces and moments

Abaqus can provide output for structural elements in terms of forces and moments acting on the cross-section at a given point. These section forces and moments are defined in the local beam coordinate system. Toggle off the rendering of beam profiles, then contour the section moment about the beam 1-axis in the elements in display group `MainA`. For clarity, reset the view so that the elements are displayed in the 1–2 plane.

To create a “bending moment”-type contour plot:

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Primary**.
2. From the list of output variables in the center of the toolbar, select **SM**.

   Abaqus/CAE automatically selects **SM1**, the first component name in the list on the right side of the **Field Output** toolbar, and displays a contour plot of the bending moment about the beam 1-axis on the deformed model shape. The deformation scale factor is chosen automatically since geometric nonlinearity was not considered in the analysis.
3. Open the **Common Plot Options** dialog box, and select a **Uniform** deformation scale factor of `1.0`.

   Color contour plots of this type typically are not very useful for one-dimensional elements such as beams. A more useful plot is a “bending moment”-type plot, which you can produce using the contour options.
4. From the main menu bar, select **Options->Contour**; or use the **Contour Options** tool in the toolbox.

   The **Contour Plot Options** dialog box appears; by default, the **Basic** tab is selected.
5. In the **Contour Type** field, toggle on **Show tick marks for line elements**.
6. Click **OK**.

   The plot shown in *Figure 4* appears. The magnitude of the variable at each node is now indicated by the position at which the contour curve intersects a “tick mark” drawn perpendicular to the element. This “bending moment”-type plot can be used for any variable (not just bending moments) for any one-dimensional element, including trusses and axisymmetric shells as well as beams.

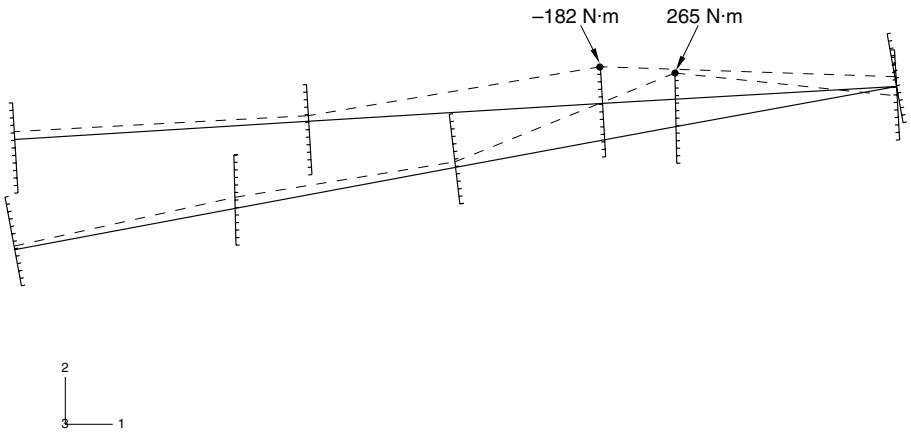


Figure 4: Bending moment diagram (moment about beam 1-axis) for elements in display group `MainA`. The locations with the highest stress (created by the bending of the elements) are indicated.

Related Abaqus examples

Related Abaqus examples are available.

- *Detroit Edison pipe whip experiment*
- *Buckling analysis of beams*
- *Crash simulation of a motor vehicle*
- *Geometrically nonlinear analysis of a cantilever beam*

Suggested reading

This section lists texts and papers that are recommended reading for obtaining additional information.

Basic beam theory

- Timoshenko, S., *Strength of Materials: Part II*, Krieger Publishing Co., 1958.
- Oden, J. T. and E. A. Ripperger, *Mechanics of Elastic Structures*, McGraw-Hill, 1981.

Basic computational beam theory

- Cook, R. D., D. S. Malkus, and M. E. Plesha, *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, 1989.
- Hughes, T. J. R., *The Finite Element Method*, Prentice-Hall Inc., 1987.

Summary

This section presents a summary of the concepts.

- The behavior of beam elements can be determined by numerical integration of the section or can be given directly in terms of area, moments of inertia, and torsional constant.
- When defining beam cross-section properties numerically, you can have the section properties calculated once at the beginning of the analysis (linear elastic material behavior is assumed) or throughout the analysis (linear or nonlinear material behavior is permitted).
- Abaqus includes a number of standard cross-section shapes. Other shapes, provided they are “thin-walled,” can be modeled using the ARBITRARY cross-section.
- The orientation of the cross-section must be defined either by specifying a third node or by defining a normal vector as part of the element property definition. The normals can be plotted in the Visualization module of Abaqus/CAE.
- The beam cross-section can be offset from the nodes that define the beam. This procedure is useful in modeling stiffeners on shells.
- The linear and quadratic beams include the effects of shear deformation. The cubic beams in Abaqus/Standard do not account for shear flexibility. The open-section beam elements in Abaqus/Standard correctly model the effects of torsion and warping (including warping constraints) in thin-walled, open sections.
- Multi-point constraints, constraint equations, and connectors can be used to connect degrees of freedom at nodes to model pinned connections, rigid links, etc.
- “Bending moment”-type plots allow the results of one-dimensional elements, such as beams, to be visualized easily.
- Display options allow you to render beam profiles for an enhanced graphical representation of the model.
- Hard copies of Abaqus/CAE plots can be obtained in PostScript (PS), Encapsulated PostScript (EPS), Tag Image File Format (TIFF), Portable Network Graphics (PNG), and Scalable Vector Graphics (SVG) formats.

Linear Dynamics

A static analysis is sufficient if you are interested in the long-term response of a structure to applied loads. However, if the duration of the applied load is short (such as in an earthquake) or if the loading is dynamic in nature (such as that from rotating machinery), you must perform a dynamic analysis.
This chapter discusses linear dynamic analysis in Abaqus/Standard; see *Nonlinear Explicit Dynamics* for a discussion of nonlinear dynamic analysis in Abaqus/Explicit.

In this section:

- *Introduction*
- *Damping*
- *Element selection*
- *Mesh design for dynamics*
- *Example: cargo crane under dynamic loading*
- *Effect of the number of modes*
- *Effect of damping*
- *Comparison with direct time integration*
- *Other dynamic procedures*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Introduction

A dynamic simulation is one in which inertia forces are included in the dynamic equation of equilibrium:

$$M\ddot{u} + I - P = 0,$$

where

M

is the mass of the structure,

$\ddot{u}$

is the acceleration of the structure,

I

are the internal forces in the structure, and

P

are the applied external forces.

The expression in the equation shown above is nothing more than Newton's second law of motion ($F = ma$).

The inclusion of the inertial forces ($M\ddot{u}$) in the equation of equilibrium is the major difference between static and dynamic analyses. Another difference between the two types of simulations is in the definition of the internal forces, *I*. In a static analysis the internal forces arise only from the deformation of the structure; in a dynamic analysis the internal forces contain contributions created by both the motion (i.e., damping) and the deformation of the structure.

In this section:

- *Natural frequencies and mode shapes*
- *Modal superposition*

Natural frequencies and mode shapes

The simplest dynamic problem is that of a mass oscillating on a spring, as shown in *Figure 1*.

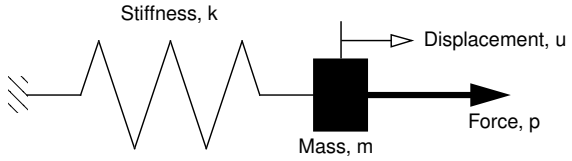


Figure 1: Mass-spring system.

The internal force in the spring is given by ku so that its dynamic equation of motion is

$$m\ddot{u} + ku - p = 0.$$

This mass-spring system has a *natural frequency* (in radians/time) given by

$$\omega = \sqrt{\frac{k}{m}}.$$

If the mass is moved and then released, it will oscillate at this frequency. If the force is applied at this frequency, the amplitude of the displacement will increase dramatically—a phenomenon known as resonance.

Real structures have a large number of natural frequencies. It is important to design structures in such a way that the frequencies at which they may be loaded are not close to the natural frequencies. The natural frequencies can be determined by considering the dynamic response of the unloaded structure ($P = 0$ in the dynamic equilibrium equation). The equation of motion is then

$$M\ddot{u} + I = 0.$$

For an undamped system $I = Ku$, so

$$M\ddot{u} + Ku = 0.$$

Solutions to this equation have the form

$$u = \phi e^{i\omega t}.$$

Substituting this into the equation of motion yields the *eigenvalue* problem

$$K\phi = \lambda M\phi,$$

where $\lambda = \omega^2$.

This system has n eigenvalues, where n is the number of degrees of freedom in the finite element model. Let λ_j be the jth eigenvalue. Its square root, ω_j, is the *natural frequency* of the jth mode of the structure, and ϕ_j is the corresponding jth *eigenvector*. The eigenvector is also known as the *mode shape* because it is the deformed shape of the structure as it vibrates in the jth mode.

The frequency extraction procedure in Abaqus/Standard is used to determine the modes and frequencies of the structure. This procedure is easy to use in that you need only specify the number of modes required or the maximum frequency of interest.

Modal superposition

The natural frequencies and mode shapes of a structure can be used to characterize its dynamic response to loads in the linear regime. The deformation of the structure can be calculated from a combination of the mode shapes of the structure using the *modal superposition* technique. Each mode shape is multiplied by a scale factor. The vector of displacements in the model, u, is defined as

$$u = \sum_{i=1}^{\infty} \alpha_i \phi_i,$$

where α_i is the modal displacement and ϕ_i is the generalized coordinate of mode i. This technique is valid only for simulations with small displacements, linear elastic materials, and no contact conditions—in other words, linear problems.

In structural dynamic problems the response of a structure usually is dominated by a relatively small number of modes, making modal superposition a particularly efficient method for calculating the response of such systems. Consider a model containing 10,000 degrees of freedom. Direct integration of the dynamic equations of motion would require the solution of 10,000 simultaneous equations at each point in time. If the structural response is characterized by 100 modes, only 100 equations need to be solved every time increment. Moreover, the modal equations are uncoupled, whereas the original equations of motion are coupled. There is an initial cost in calculating the modes and frequencies, but the savings obtained in the calculation of the response greatly outweigh the cost.

If nonlinearities are present in the simulation, the natural frequencies may change significantly during the analysis, and modal superposition cannot be employed. In this case direct integration of the dynamic equation of equilibrium is required, which is much more expensive than modal analysis.

A problem should have the following characteristics for it to be suitable for linear transient dynamic analysis:

- The system should be linear: linear material behavior, no contact conditions, and no nonlinear geometric effects.
- The response should be dominated by relatively few frequencies. As the frequency content of the response increases, such as is the case in shock and impact problems, the modal superposition technique becomes less effective.
- The dominant loading frequencies should be in the range of the extracted frequencies to ensure that the loads can be described accurately.
- The initial accelerations generated by any suddenly applied loads should be described accurately by the eigenmodes.
- The system should not be heavily damped.

Damping

If an undamped structure is allowed to vibrate freely, the magnitude of the oscillation is constant. In reality, however, energy is dissipated by the structure's motion, and the magnitude of the oscillation decreases until the oscillation stops. This energy dissipation is known as damping.

Damping is usually assumed to be viscous or proportional to velocity. The dynamic equilibrium equation can be rewritten to include damping as

$$M\ddot{u} + I - P = 0,$$
$$I = Ku + C\dot{u},$$

where

C

is the damping matrix for the structure and

$\dot{u}$

is the velocity of the structure.

The dissipation of energy is caused by a number of effects, including friction at the joints of the structure and localized material hysteresis. Damping is a convenient way of including the important absorption of energy without modeling the effects in detail.

In Abaqus/Standard the eigenmodes are calculated for the undamped system, yet most engineering problems involve some kind of damping, however small. The relationship between the damped natural frequency and the undamped natural frequency for each mode is

$$\omega_d = \omega\sqrt{1 - \xi^2},$$

where

ω_d

is the damped eigenvalue,

$\xi = \frac{c}{c_{cr}}$

is the damping ratio, which is the fraction of critical damping,

c

is the damping of that mode shape, and

c_{cr}

is the critical damping.

The eigenfrequencies of the damped system are very close to the corresponding quantities for the undamped system for small values of ξ ($\xi < 0.1$). As ξ increases, the undamped eigenfrequencies become less accurate; and as ξ approaches 1, the use of undamped eigenfrequencies becomes invalid.

If a structure is critically damped ($\xi = 1$), after any disturbance it will return to its initial static configuration as quickly as possible without overshooting (*Figure 1*).

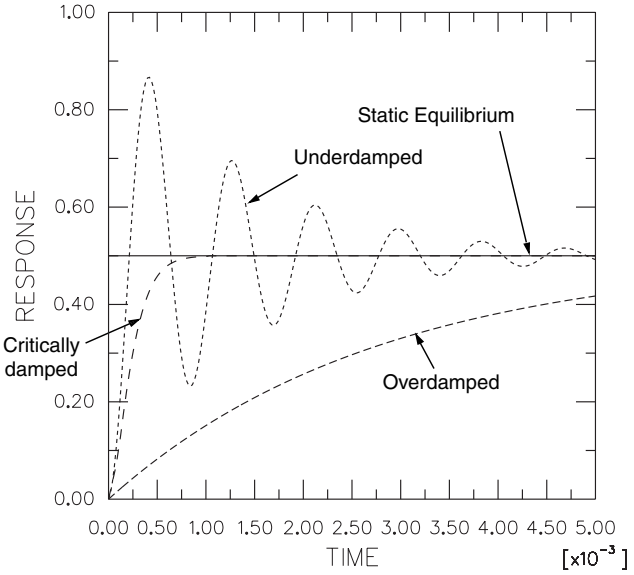


Figure 1: Damped motion patterns for various values of ξ.

Definition of damping in Abaqus/Standard

In Abaqus/Standard a number of different types of damping can be defined for a transient modal analysis: direct modal damping, Rayleigh damping, and composite modal damping.

Damping is defined for modal dynamic procedures. The damping is part of the step definition, and different amounts of damping can be defined for each mode.

Direct modal damping

The fraction of critical damping, ξ, associated with each mode can be defined using direct modal damping. Typically, values in the range of 1% to 10% of critical damping are used. Direct modal damping allows you to define precisely the damping of each mode of the system.

Rayleigh damping

In Rayleigh damping the assumption is made that the damping matrix is a linear combination of the mass and stiffness matrices,

$$C = \alpha M + \beta K,$$

where α and β are user-defined constants. Although the assumption that the damping is proportional to the mass and stiffness matrices has no rigorous physical basis, in practice the damping distribution rarely is known in sufficient detail to warrant any other more complicated model. In general, this model ceases to be reliable for heavily damped systems; that is, above approximately 10% of critical damping. As with the other forms of damping, you can define precisely the Rayleigh damping of each mode of the system.

For a given mode i, the damping ratio, ξ_i, and the Rayleigh damping values, α and β, are related through

$$\xi_i = \frac{\alpha}{2\omega_i} + \frac{\beta\omega_i}{2}.$$

Composite damping

In composite damping a fraction of critical damping is defined for each material, and a composite damping value is found for the whole structure. This option is useful when many different materials are present in the structure. Composite damping is not discussed further in this guide.

Choosing damping values

In most linear dynamic problems the proper specification of damping is important to obtain accurate results. However, damping is approximate in the sense that it models the energy absorbing characteristics of the structure without attempting to model the physical mechanisms that cause them. Therefore, it is difficult to determine the damping data required for a simulation. Occasionally, you may have data available from dynamic tests, but often you will have to work with data gleaned from references or experience. In such cases you should be very cautious in interpreting the results, and you should use parametric studies to assess the sensitivity of the simulation to damping values.

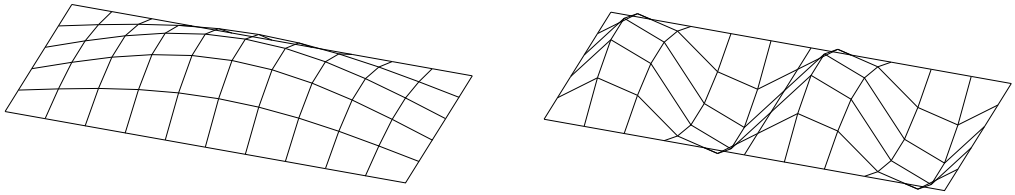
Element selection

Virtually all of the elements in Abaqus can be used in dynamic analyses. In general the rules for selecting the elements are the same as those for static simulations. However, for simulations of impact and blast loading, first-order elements should be used. They have a lumped mass formulation, which is better able to model the effect of stress waves than the consistent mass formulation used in the second-order elements.

Mesh design for dynamics

When you are designing meshes for dynamic simulations, you need to consider the mode shapes that will be excited in the response and use a mesh that is able to represent those mode shapes adequately. This means that a mesh that is adequate for a static simulation may be unsuitable for calculating the dynamic response to loading that excites high frequency modes.

Consider, for example, the plate shown in *Figure 1*. The mesh of first-order shell elements is adequate for a static analysis of the plate under a uniform load and is also suitable for the prediction of the first mode shape. However, the mesh is clearly too coarse to be able to model the sixth mode accurately.

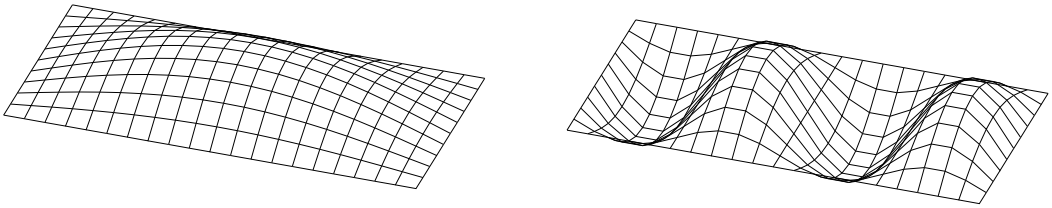


(a) Mode 1: 31.1 Hz

(b) Mode 6: 140 Hz

Figure 1: Vibration frequencies and corresponding mode shapes of the plate based on the coarse mesh.

Figure 2 shows the same plate modeled with a refined mesh of first-order elements. The displaced shape for the sixth mode now looks much better, and the frequency predicted for this mode is more accurate. If the dynamic loading on the plate is such that there is significant excitation of this mode, the refined mesh must be used; the results from the coarse mesh will not be accurate.



(a) Mode 1: 30.2 Hz

(b) Mode 6: 124 Hz

Figure 2: Vibration frequencies and corresponding mode shapes of the plate based on the fine mesh.

Example: cargo crane under dynamic loading

This example models a light-service, cargo crane under dynamic loading.

In this section:

- *Modifications to the model*
- *Results*
- *Postprocessing*

Modifications to the model

This example expands the previous cargo crane analysis.
Starting with the analysis in *Example: cargo crane*, you have now been asked to investigate what happens when a load of 10 kN is dropped onto the lifting hook for 0.2 seconds.

The connections at points *A*, *B*, *C*, and *D* (see *Figure 1*) can only withstand a maximum pull-out force of 100 kN. You have to decide whether or not any of these connections will break.

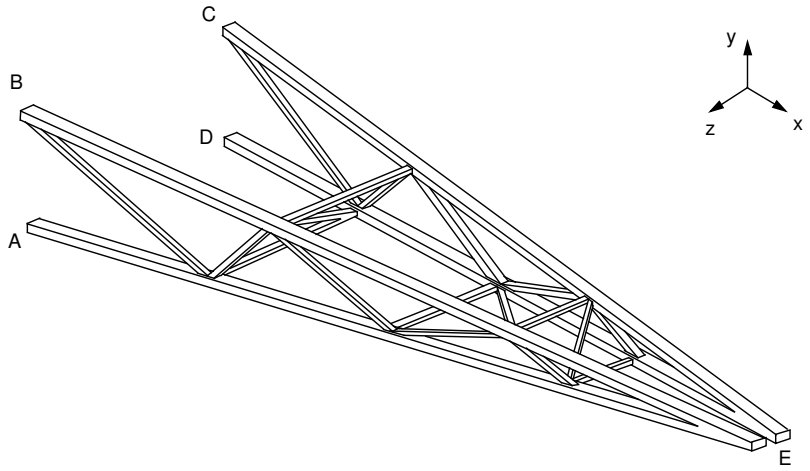


Figure 1: Cargo crane.

The short duration of the loading means that inertia effects are likely to be important, making dynamic analysis essential. You are not given any information regarding the damping of the structure. Since there are bolted connections between the trusses and the cross bracing, the energy absorption caused by frictional effects is likely to be significant. Based on experience, you therefore choose 5% of critical damping in each mode.

The magnitude of the applied load versus time is shown in *Figure 2*.

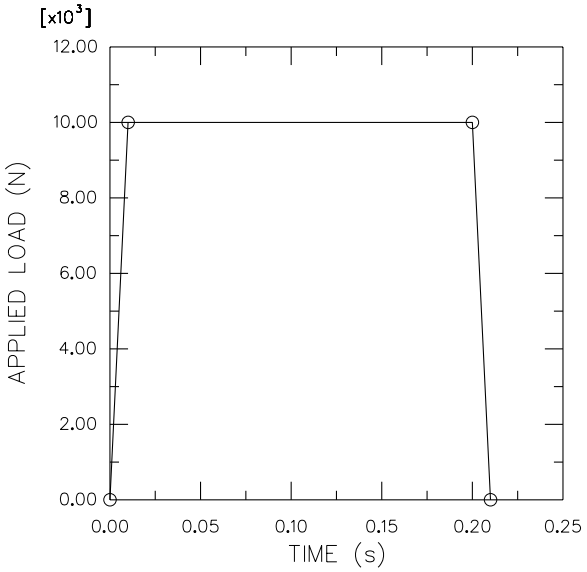


Figure 2: Load-time characteristic.

Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Cargo crane – dynamic loading*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Cargo crane dynamic loading**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Open the model database file `Crane.cae`, and copy the `Static` model to a model named `Dynamic`. The dynamic analysis model is basically the same as the static analysis model, except for the modifications described below.

Material

In dynamic simulations the density of every material must be specified so that the mass matrix can be formed. The steel in the crane has a density of 7800 kg/m^3.

In this model the material properties were specified as part of the section definition. Thus, you will need to edit the `BracingSection` and `MainMemberSection` section definitions to specify the density. In the **Specify section material density** field of the **Edit Beam Section** dialog box, enter a value of `7800` for each section definition.



Note: If material data are defined independently of the section properties, the density is included by editing the material definition and selecting **General->Density** in the **Edit Material** dialog box.

Steps

The step definitions that are used for the dynamic analysis are substantially different from those used in the static analysis. Therefore, the static step created previously will be replaced by two new steps.

The first step in the dynamic analysis calculates the natural frequencies and mode shapes of the structure. The second step then uses these data to calculate the transient modal dynamic response of the cargo crane. In this analysis we will assume that everything is linear. If you want to model any nonlinearities in this simulation, direct integration of the equations of motion using the implicit dynamic procedure must be performed instead. See *Nonlinear dynamics* for further details.

Abaqus/Standard offers three eigenvalue extraction methods: Lanczos, automatic multi-level substructuring (AMS), and subspace iteration. Both the Lanczos and AMS methods are generally faster than subspace iteration when a large number of eigenmodes is required for a system with many degrees of freedom. While the AMS solver is more efficient for very large systems, the Lanczos solver has the most general capabilities. The subspace iteration method may be faster when only a few (less than 20) eigenmodes are needed.

We use the Lanczos eigensolver in this analysis and request 30 eigenvalues. Instead of specifying the number of modes required, it is also possible to specify the minimum and maximum frequencies of interest so that the step will complete once Abaqus/Standard has found all of the eigenvalues inside the specified range. A shift point may also be specified so that eigenvalues nearest the shift point will be extracted. By default, no minimum or maximum frequency or shift is used. If the structure is not constrained against rigid body modes, the shift value should be set to a small negative value to remove numerical problems associated with rigid body motion.

To replace the static step with a frequency extraction step:

1. In the Model Tree, expand the **Steps** container. Then, click mouse button 3 on the step named **Tip Load** and select **Replace** from the menu that appears. In the **Replace Step** dialog box, select **Frequency** from the list of available **Linear perturbation** procedures.

Model attributes that cannot be converted will be deleted. In this case the concentrated loads are deleted because they cannot be used in a frequency extraction step. However, the boundary conditions and output requests associated with the static step are inherited by the frequency extraction step.

2. In the **Basic** tabbed page of the **Edit Step** dialog box, enter the step description `First 30 modes`; accept the Lanczos eigensolver option; and request 30 eigenvalues.
3. Rename the step to `Extract Frequencies` by clicking mouse button 3 on the name `Tip Load` and selecting **Rename** from the menu that appears.

In structural dynamic analysis the response is usually associated with the lower modes. However, enough modes should be extracted to provide a good representation of the dynamic response of the structure. One way of checking that a sufficient number of eigenvalues has been extracted is to look at the total effective mass in each degree of freedom, which indicates how much of the mass is active in each direction of the extracted modes. The effective masses are tabulated in the data file under the eigenvalue output. Ideally, the sum of the modal effective masses for each mode in each direction should be at least 90% of the total mass. See *Effect of the number of modes* for more information.

The modal dynamics procedure will be used to perform the transient dynamic analysis. The transient response will be based on all the modes extracted in the first analysis step; 5% of critical damping should be used in all 30 modes.

To create a transient modal dynamics step:

1. In the Model Tree, double-click the **Steps** container to create a new step. Select **Modal dynamics** from the list of available **Linear perturbation** procedures, and name the step `Transient modal dynamics`. Insert the step after the frequency extraction step defined above.
2. In the **Basic** tabbed page of the **Edit Step** dialog box, enter the description `Crane Response to Dropped Load` and specify a time period of `0.5` and a time increment of `0.005`. In dynamic analysis time is a real, physical quantity.
3. In the **Damping** tabbed page of the **Edit Step** dialog box, specify direct modal damping and enter a critical damping fraction of `0.05` for modes `1` through `30`.

Output

Using the **Field Output Requests Manager**, modify the field output requests for the `Extract Frequencies` step so that the **Preselected defaults** are selected. By default, Abaqus/Standard writes the mode shapes to the output database (`.odb`) file so that they can be plotted using the Visualization module. The nodal displacements for each mode shape are normalized so that the maximum displacement is unity. Thus, these results, and the corresponding stresses and strains, are not physically meaningful: they should be used only for relative comparisons.

Dynamic analyses usually require many more increments than static analyses to complete. As a consequence, the volume of output from dynamic analyses can be very large, and you should control the output requests to keep the output files to a reasonable size. In this example you will request output of the deformed shape to the output database file at the end of every fifth increment. There will be 100 increments in the step (0.5/0.005); therefore, there will be 20 frames of field output.

In addition, you will write the displacements at the loaded end of the model (for example, set `Tip-a`) and the reaction forces at the fixed end (set `Attach`) as history data to the output database file every increment so that a higher resolution of these data will be available. In dynamic analyses we are also concerned about the energy distribution in the model and what form the energy takes. Kinetic energy is present in the model as a result of the motion of the mass; strain energy is present as a result of the displacement of the structure; energy is also dissipated through damping. By default, whole model energies are written as history data to the `.odb` file for

the modal dynamic procedure. For this analysis you will restrict the energy output to the kinetic, internal, and viscous dissipation energies.

To request output for the transient modal dynamics analysis step:

1. Open the **Field Output Requests Manager**. Select the cell labeled `Created` that appears in the column labeled **Transient modal dynamics** (you may need to enlarge the column to see the complete step name).
2. Edit the field output request so that only the nodal displacements are written to the `.odb` file every `5` increments.
3. Open the **History Output Requests Manager**. Edit the default output request so that only ALLIE, ALLKE, and ALLVD are written after every increment. In addition, create two new output requests in the step labeled `Transient modal dynamics`. In the first write the displacements (translations only) for the set `Tip-a` after every increment; in the second, the reaction forces (not the moments) for the set `Attach` after every increment.

Loads and boundary conditions

The boundary conditions are the same as for the static analysis. Since these were retained during the step replacement operation, no new boundary conditions need to be defined.

Apply a concentrated force to the tip of the crane. The magnitude of this load is time dependent, as illustrated in *Figure 2*. The time dependence of the load can be defined using an amplitude curve. The actual magnitude of the load applied at any point in time is obtained by multiplying the magnitude of the load (−10,000 N) by the value of the amplitude curve at that time.

To specify a time-dependent load:

1. Begin by defining an amplitude curve. In the Model Tree, double-click the **Amplitudes** container. Name the amplitude `Bounce`, and choose the type **Tabular**. Enter the data shown in *Table 1* in the **Edit Amplitude** dialog box. Accept the default selection of **Step time** as the time span, and specify `0.25` as the smoothing parameter value.

   Tip: Use mouse button 3 to access the table options.

   Table 1: Amplitude curve data.

Time (sec)	Amplitude
0.0	0.0
0.01	1.0
0.2	1.0
0.21	0.0

2. Now define the load. In the Model Tree, double-click the **Loads** container. Apply the load in the `Transient modal dynamics` step, name the load `Dyn load`, and choose **Concentrated force** as the load type. Apply the load to set `Tip-b`. The equation constraint defined previously between sets `Tip-a` and `Tip-b` means that the load will be carried equally by both halves of the crane.
3. In the **Edit Load** dialog box, enter a value for **CF2** of `-1.E4`, and choose `Bounce` for the amplitude.

In this example the structure has no initial velocities or accelerations, which is the default. However, if you wanted to define initial velocities, you could do so by selecting **Predefined Field->Create** from the main menu

bar and assigning initial translational velocities to selected regions of the model in the initial step. You would also have to edit the definition of the modal dynamics step to use initial conditions.

Running the analysis

Create a job named `DynCrane` with the following description: `3-D model of light-service cargo crane dynamic analysis`.

Save your model in a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected and investigate the cause of any warning messages, taking action as necessary.

Results

You can use the **Job Monitor** to monitor information concerning your submitted job.

The **Job Monitor** gives a brief summary of the automatic time incrementation used in the analysis for each increment. The information is written as soon as the increment is completed, so that you can monitor the analysis as it is running. This facility is useful in large, complex problems. The information given in the **Job Monitor** is the same as that given in the status file (`DynCrane.sta`).

Examine the **Job Monitor** and the printed output data file (`DynCrane.dat`) to evaluate the analysis results.

Job monitor

In the **Job Monitor**, the first column shows the step number and the second column gives the increment number. The sixth column shows the number of iterations Abaqus/Standard needed to obtain a converged solution in each increment. Looking at the contents of the **Job Monitor**, we can see that the time increment associated with the single increment in Step 1 is very small. This step uses no time, because time is not relevant in a frequency extraction procedure.

The output for Step 2 shows that the time increment size is constant throughout the step and that each increment requires only one iteration. The bottom of the **Job Monitor** is shown in *Figure 1*.

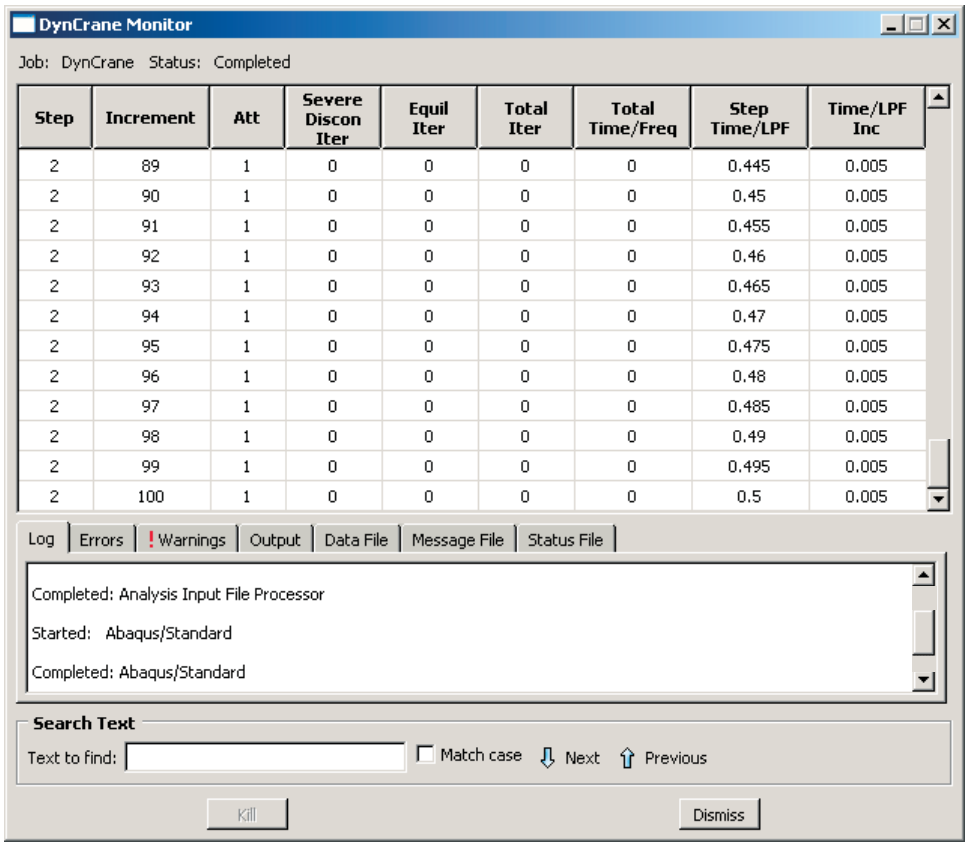


Figure 1: Bottom portion of the Job Monitor: cargo crane dynamic analysis.

Data file

Click the **Data File** tab in the **Job Monitor** to display the data file in a tabbed page at the bottom of the dialog box. The primary results for Step 1 are the extracted eigenvalues, participation factors, and effective mass, as shown below.

```
                              E I G E N V A L U E    O U T P U T

MODE NO    EIGENVALUE       FREQUENCY          GENERALIZED MASS  COMPOSITE
MODAL DAMPING
                         (RAD/TIME)  (CYCLES/TIME)


      1       1773.4     42.112     6.7023       151.92           0.0000

      2       7016.8     83.766     13.332       30.206           0.0000

      3       7644.1     87.431     13.915       90.400           0.0000

      4       22999.     151.65     24.136       250.64           0.0000

      5       24714.     157.21     25.020       275.88           0.0000

      6       34811.     186.58     29.695       493.15           0.0000

      7       42748.     206.76     32.906       1106.4           0.0000

      8       46473.     215.58     34.310       86.173           0.0000

      9       47446.     217.82     34.667       2577.2           0.0000

     10       56050.     236.75     37.680       3569.2           0.0000

....
     25    2.26885E+05   476.32     75.809       207.46           0.0000

     26    2.42798E+05   492.75     78.423       127.02           0.0000

     27    2.84057E+05   532.97     84.825       1240.8           0.0000

     28    2.92450E+05   540.79     86.069       330.74           0.0000

     29    3.13943E+05   560.31     89.176       272.39           0.0000

     30    3.64774E+05   603.97     96.124       64.971           0.0000
```

The highest frequency extracted is 96 Hz. The period associated with this frequency is 0.0104 seconds, which is comparable to the fixed time increment of 0.005 seconds. There is no point in extracting modes whose period is substantially smaller than the time increment used. Conversely, the time increment must be capable of resolving the highest frequencies of interest.

The column for generalized mass lists the mass of a single degree of freedom system associated with that mode.

The table of participation factors indicates the predominant degrees of freedom in which the modes act. The results indicate, for example, that mode 1 acts predominantly in the 3-direction.

```
          P A R T I C I P A T I O N   F A C T O R S

 MODE NO    X-COMPONENT    Y-COMPONENT    Z-COMPONENT    X-ROTATION
```

```
Y-ROTATION     Z-ROTATION

       1     -6.11696E-04   -6.14521E-03     1.4284         0.71335
-6.0252        -3.37773E-02
       2      0.18470       -0.25677        8.31954E-04    1.68388E-03
-6.05012E-03     -1.6826
       3     -0.17440         1.5515        4.88123E-03   -8.04039E-03
3.24495E-02      9.2746
       4     -8.68256E-05   -9.61259E-03    8.23615E-02    0.21604
 1.2334        -2.97905E-02
       5     -3.80675E-03    1.13829E-03   -3.04304E-02   -0.59220
 1.7593        -2.20144E-02
       6      3.71618E-02   -0.35674        6.05241E-03   -1.67946E-02
6.71292E-03    -0.96432
       7     -2.48508E-03   -1.58332E-03    6.19821E-02    5.09235E-02
-0.29901        -6.65849E-04
       8     -7.03851E-02    2.31655E-02    0.72459        0.49275
-3.8778          6.69085E-02
       9      3.59820E-02   -2.34811E-02    2.23695E-02    1.47243E-02
-0.12808         6.65955E-04
      10      3.48679E-02    4.02884E-02    1.96398E-02    1.09545E-02
-6.84066E-02     3.72037E-02
....
      25     -8.25375E-02   -0.22218       -3.54545E-02    3.39238E-02
-2.18245E-02    -0.18688
      26     -1.98905E-02   -0.35111        4.61269E-02   -2.12563E-02
-1.27532E-02    -0.18939
      27      1.71772E-02    2.51340E-02    2.26524E-02   -1.02593E-02
-4.31559E-02     2.78870E-02
      28      4.73352E-02    2.79265E-02   -0.11860        5.19825E-02
0.24175        -1.12541E-04
      29      9.83488E-03   -3.64823E-03    4.65504E-03   -3.17284E-03
-1.56708E-02    -2.82848E-03
      30      4.83733E-02    1.85495E-02    0.13426       -2.21861E-02
-0.35882        -1.87612E-02
```

The table of effective mass indicates the amount of mass active in each degree of freedom for any one mode. The results indicate that the first mode with significant mass in the 2-direction is mode 3. The total modal effective mass in the 2-direction is 378.26 kg.

```
                    E F F E C T I V E   M A S S

 MODE NO    X-COMPONENT    Y-COMPONENT    Z-COMPONENT    X-ROTATION
Y-ROTATION     Z-ROTATION


       1      5.68458E-05    5.73721E-03     309.98         77.309
 5515.3         0.17333
       2      1.0304          1.9915        2.09072E-05    8.56481E-05
1.10567E-03      85.521
       3      2.7495          217.62        2.15392E-03    5.84420E-03
9.51888E-02      7776.2
       4      1.88952E-06    2.31599E-02     1.7002         11.699
 381.31         0.22244
       5      3.99791E-03    3.57461E-04     0.25547        96.753
 853.88         0.13370
       6      0.68104         62.759        1.80648E-02    0.13910
2.22229E-02      458.58
       7      6.83296E-03    2.77373E-03     4.2507         2.8692
 98.926         4.90544E-04
       8      0.42691        4.62440E-02     45.243         20.923
```

```
 1295.8          0.38578
       9         3.3366          1.4209          1.2896         0.55874
 42.275          1.14296E-03
      10         4.3393          5.7933          1.3767         0.42830
 16.702           4.9402
....
      25         1.4133          10.241         0.26078         0.23875
9.88154E-02       7.2457
      26       5.02526E-02       15.658         0.27026         5.73911E-02
2.06589E-02       4.5558
      27       0.36612         0.78387         0.63672         0.13060
 2.3110           0.96499
      28       0.74106         0.25794          4.6523         0.89371
 19.329           4.18897E-06
      29       2.63473E-02     3.62545E-03     5.90262E-03     2.74218E-03
6.68933E-02      2.17924E-03
      30       0.15203         2.23557E-02      1.1711         3.19804E-02
 8.3651           2.28687E-02

 TOTAL           22.198          378.26          373.69          269.78
 8348.4           8518.0
```

The total mass of the model is given earlier in the data file and is 414.34 kg.

To ensure that enough modes have been used, the total effective mass in each direction should be a large proportion of the mass of the model (say 90%). However, some of the mass of the model is associated with nodes that are constrained. This constrained mass is approximately one-quarter of the mass of all the elements attached to the constrained nodes, which, in this case, is approximately 28 kg. Therefore, the mass of the model that is able to move is 385 kg.



Tip: To determine the mass of the elements attached to the constrained nodes, switch to the Mesh module, click the query tool , and select **Mass properties** from the list of general queries. In the prompt area, select **Select mesh entities**, and select the six elements attached to the constrained nodes. The total mass is displayed in the message area (114 kg).

The effective mass in the *x*-, *y*-, and *z*-directions is 6%, 98%, and 97%, respectively, of the mass that can move. The total effective mass in the 2- and 3-directions is well above the 90% recommended earlier; the total effective mass in the 1-direction is much lower. However, since the loading is applied in the 2-direction, the response in the 1-direction is not significant.

The data file does not contain any results for the modal dynamics step, because all of the data file output requests were suppressed.

Postprocessing

Enter the Visualization module, and open the output database file `DynCrane.odb`.

In this section:

- *Plotting mode shapes*
- *Animation of results*
- *Determining the peak pull-out force*

Plotting mode shapes

You can visualize the deformation mode associated with a given natural frequency by plotting the mode shape associated with that frequency.

1. In the context bar, click the frame selector tool .

   The **Frame Selector** dialog box appears. Drag the bottom corner of the dialog box to enlarge it so that both step names are clearly visible.

2. Drag the frame slider to select frame `1` in the **Extract Frequencies** step. This is the first eigenmode.

3. From the main menu bar, select **Plot->Deformed Shape**; or use the tool in the toolbox.

   Abaqus/CAE displays the deformed model shape associated with the first vibration mode, as shown in *Figure 1*.

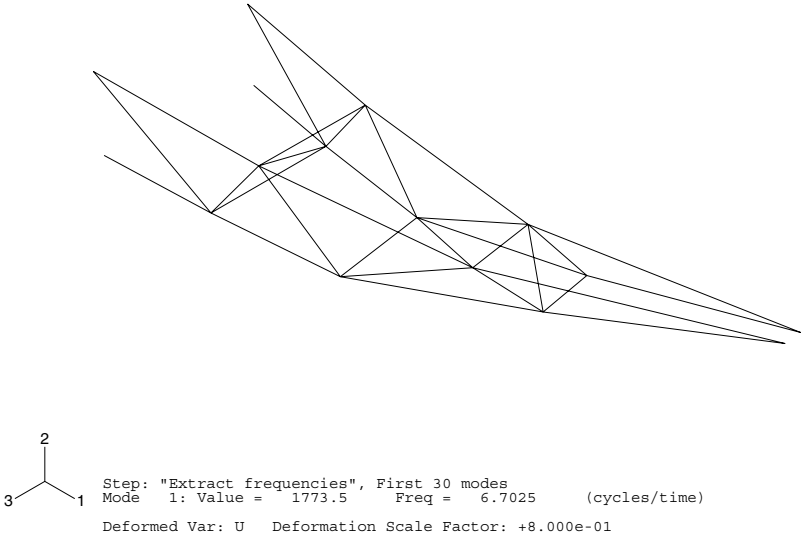
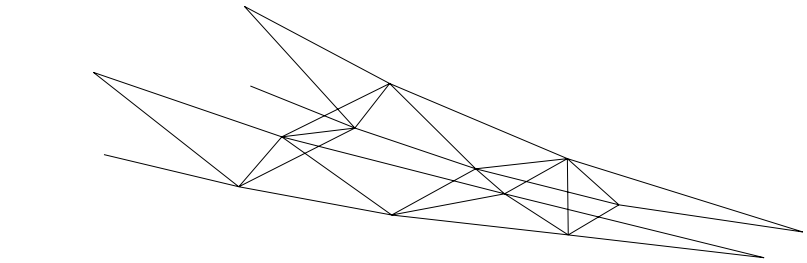


Figure 1: Mode 1.

4. Select the third mode (frame `3` in the **Extract Frequencies** step) from the **Frame Selector** dialog box. Afterward, close the dialog box.

   Abaqus/CAE displays the third mode shape shown in *Figure 2*.



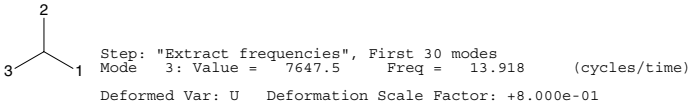


Figure 2: Mode 3.



Note:

A complete list of the available frames is given in the **Step/Frame** dialog box (**Result**->**Step/Frame**). This dialog box offers an alternative means to switching between frames.

Animation of results

You will animate the analysis results.
First create a scale factor animation of the third eigenmode. Then create a time-history animation of the transient results.

Create a scale factor animation of an eigenmode

1. From the main menu bar, select **Animate->Scale Factor**; or use the  tool in the toolbox.

   Abaqus/CAE displays the third mode shape and steps through different deformation scale factors ranging from 0 to 1.

   Abaqus/CAE also displays the movie player controls on the right side of the context bar.

2. In the context bar, click  to pause the animation.

Create a time-history animation of the transient results

1. From the main menu bar, select **Result->Active Steps/Frames** to select which frames will be active in the history animation.

   Abaqus/CAE displays the **Active Steps/Frames** dialog box.

2. Toggle the step names so that only the second step (**Transient modal dynamics**) is selected.
3. Click **OK** to accept the selection and to close the dialog box.
4. From the main menu bar, select **Animate->Time History**; or use the  tool from the toolbox.

   Abaqus/CAE steps through each available frame of the second step. The state block indicates the current step and increment throughout the animation. After the last increment of this step is reached, the animation process repeats itself.

5. You can customize the deformed shape plot while the animation is running.
   a. Display the **Common Plot Options** dialog box.
   b. Choose **Uniform** from the **Deformation Scale Factor** field.
   c. Enter `15.0` as the deformation scale factor value.
   d. Click **Apply** to apply your change.

      Abaqus/CAE now steps through the frames in the second step with a deformation scale factor of `15.0`.

   e. Choose **Auto-compute** from the **Deformation Scale Factor** field.
   f. Click **OK** to apply your change and to close the **Common Plot Options** dialog box.

      Abaqus/CAE now steps through the frames in the second step with a default deformation scale factor of `0.8`.

Determining the peak pull-out force

To find the peak pull-out force at the attachment points, create an *X–Y* plot of the reaction force in the 1-direction (variable RF1) at the attached nodes.
This involves plotting multiple curves at the same time.

Plot multiple curves

1. In the Results Tree, click mouse button 3 on **History Output** for the output database named `DynCrane.odb`. From the menu that appears, select **Filter**.
2. In the filter field, enter `*RF1*` to restrict the history output to just the reaction force components in the 1-direction.
3. From the list of available history output, select the four curves (using **[Ctrl] + Click**) that have the following form:

   ```
   Reaction Force: RF1 PI: TRUSS-1 Node xxx in NSET ATTACH
   ```

4. Click mouse button 3, and select **Plot** from the menu that appears.

   Abaqus/CAE displays the selected curves.

5. Click  in the prompt area to cancel the current procedure.

Position the grid

1. Double-click the plot to open the **Chart Options** dialog box.
2. In this dialog box, switch to the **Grid Area** tabbed page.
3. In the **Size** region of this page, select the **Square** option.
4. Use the slider to set the size to **75**.
5. In the **Position** region of this page, select the **Auto-align** option.
6. From the available alignment options, select the last one (position the grid in the lower right corner of the viewport).
7. Click **Dismiss**.

Position the legend

1. Double-click the legend to open the **Chart Legend Options** dialog box.
2. In this dialog box, switch to the **Area** tabbed page.
3. In the **Position** region of this page, toggle on **Inset**.
4. To display the minimum and maximum values in the legend, switch to the **Contents** tabbed page of the dialog box. In the **Numbers** region of this page, toggle on **Show min/max**.
5. Click **Dismiss**.
6. Drag the legend in the viewport to reposition it.

   The resulting plot (which has been customized) is shown in *Figure 1*. The curves for the two nodes at the top of each truss (points B and C) are almost a reflection of those for the nodes on the bottom of each truss (points A and D).



Note: To modify the curve styles, click  in the Visualization toolbox to open the **Curve Options** dialog box.

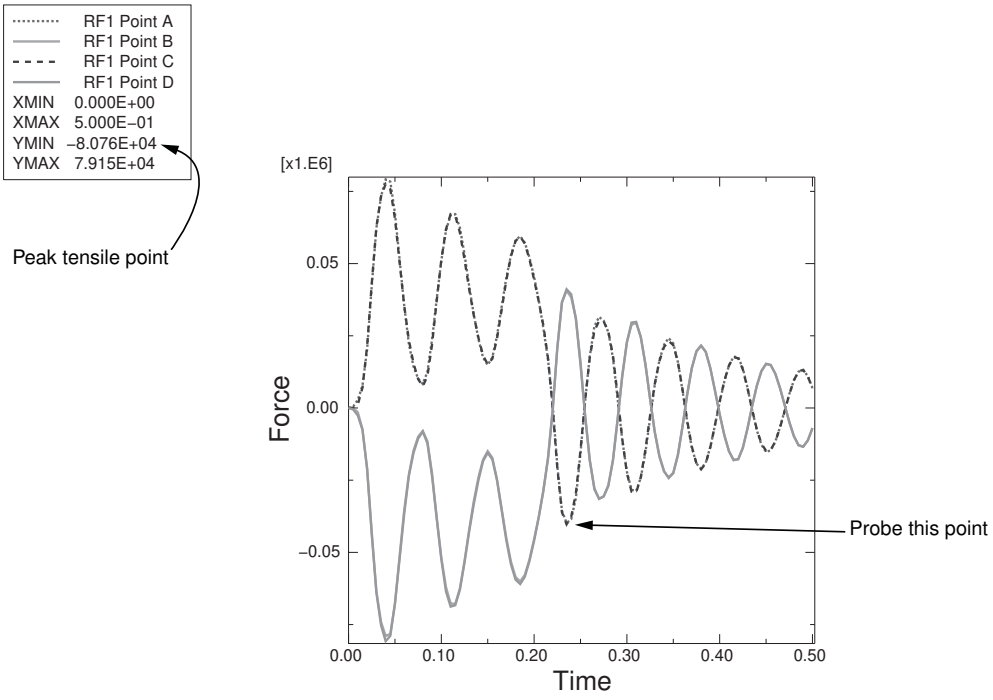


Figure 1: History of the reaction forces at the attached nodes.

Query the X–Y plot

At the attachment points at the top of each truss structure, the peak tensile force is around 80 kN, which is below the 100 kN capacity of the connection. Keep in mind that a negative reaction force in the 1-direction means that the member is being pulled away from the wall. The lower attachments are in compression (positive reaction force) while the load is applied but oscillate between tension and compression after the load has been removed. The peak tensile force is about 40 kN, well below the allowable value. To find this value, probe the *X–Y* plot.

1. From the main menu bar, select **Tools->Query**.

   The **Query** dialog box appears.

2. Click **Probe Values** in the **Visualization Module Queries** field.

   The **Probe Values** dialog box appears.

3. Select the point indicated in *Figure 1*.

   The *Y*-coordinate of this point is –40.30 kN, which corresponds to the value of the reaction force in the 1-direction.

Effect of the number of modes

For this simulation 30 modes were used to represent the dynamic behavior of the structure. The total modal effective mass for all of these modes was well over 90% of the mass of the structure that can move in the *y*- and *z*-directions, indicating that the dynamic representation is adequate.

Figure 1 shows the displacement in the direction of degree of freedom 2 of the node in set `Tip-a` versus time and illustrates the effect of using fewer modes on the quality of the results. If you look at the table of effective mass, you will see that the first significant mode in the 2-direction is mode 3, which accounts for the lack of response when only two modes are used. The displacement of this node in the direction of degree of freedom 2 for the analyses using five modes and 30 modes is similar after 0.2 seconds; however, the early response differs, suggesting that there are significant modes in the range of 5–30 relating to the early response. When five modes are used, the total modal effective mass in the 2-direction is only 57% of the moveable mass.

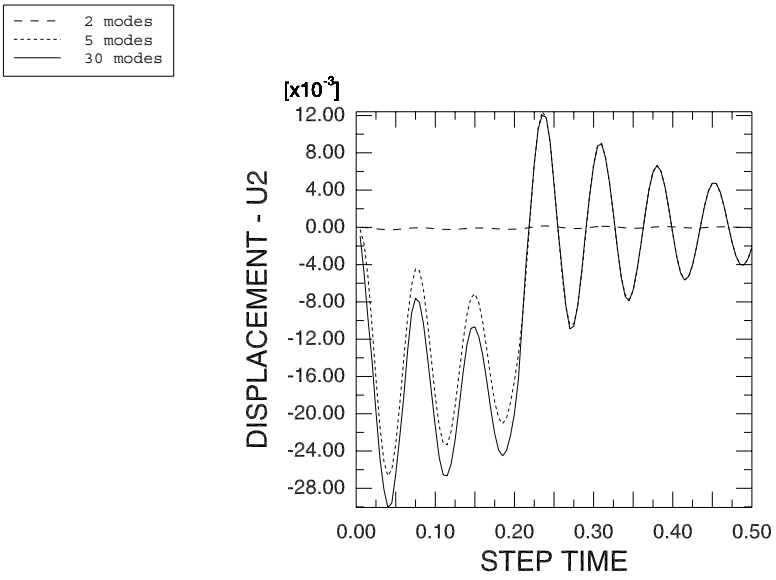


Figure 1: Effect of different numbers of modes on the results.

Effect of damping

In this simulation we used 5% of critical damping in all modes. This value was chosen from experience, based on the fact that the bolted connections between the trusses and the cross bracing might absorb significant energy as a result of local frictional effects. In cases such as this, where accurate data are not available, it is important to investigate the effect of the choices that you make.

Figure 1 compares the history of the reaction force at one of the top connections (point C) when 1%, 5%, and 10% of critical damping are used. As expected, the oscillations at lower damping levels do not diminish as quickly as those at higher damping levels, and the peak force is higher in the models with lower damping. With damping ratios even as low as 1%, the peak pull-out force is 85 kN, which is still less than the strength of the connection (100 kN). Therefore, the cargo crane should retain its integrity under this drop load.

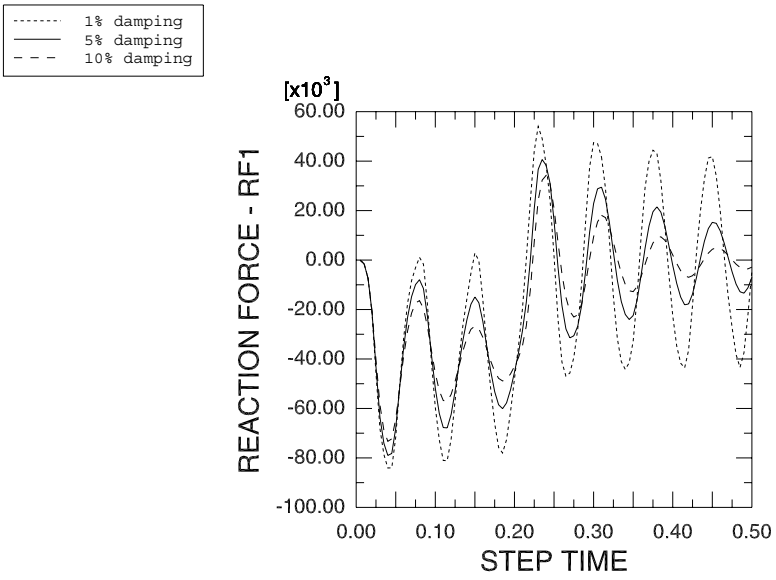


Figure 1: Effect of damping ratio on the pull-out force.

Comparison with direct time integration

Since this is a transient dynamic analysis, it is natural to consider how the results compare with those obtained using direct integration of the equations of motion. Direct integration can be performed with either implicit (Abaqus/Standard) or explicit (Abaqus/Explicit) methods. Here we extend the analysis to use the explicit dynamics procedure.

A direct comparison with the results presented earlier is not possible since the B33 element type and direct modal damping are not available in Abaqus/Explicit. Thus, in the Abaqus/Explicit analysis the element type is changed to B31 and Rayleigh damping is used in place of direct modal damping.

Copy the `Dynamic` model to one named `explicit`. All subsequent changes should be made to the `explicit` model.

1. Delete the modal dynamics step. When Abaqus/CAE warns you that deleting a step also deletes step-dependent objects, click **Yes**.
2. Replace the remaining frequency extraction step with an explicit dynamics step, and specify a time period of `0.5` s. In addition, edit the step to use linear geometry (toggle off **Nlgeom**).

   This will result in a linear analysis.
3. Rename the step to `Transient dynamics`.
4. Create two additional history output requests. In the first, request displacement history for the set `Tip-a`; in the second, request reaction force history for the set `Attach`.
5. Add mass proportional damping to the bracing section properties. To do this, double-click **BracingSection** underneath the **Sections** container in the Model Tree; in the section editor that appears, click the **Damping** tab.

   In the **Stiffness Proportional Material Damping** region, enter a value of `15` for **Alpha** and `0` for the remaining damping quantities.

   These values produce a reasonable trade-off in the values of critical damping at low and high frequencies of the structure. For the three lowest natural frequencies the effective value of ξ is greater than 0.05; but as was shown in *Figure 1*, the first two modes do not contribute significantly to the response. For the remaining modes, the value of ξ is less than 0.05. The variation of ξ as a function of natural frequency is shown in *Figure 1*.

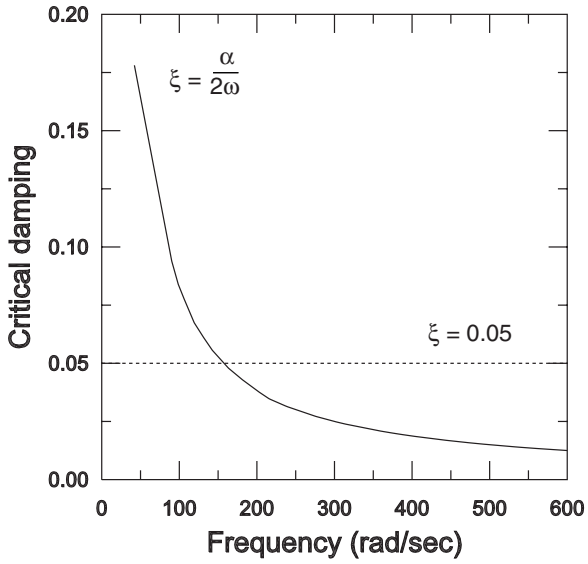


Figure 1: Variation of damping ratio with frequency corresponding to the specified Rayleigh factors (α = 15, β = 0).

6. Repeat the above step for the main member section properties.
7. Redefine the tip load at set `Tip-b`. Specify **CF2** = `-10000`, and use the amplitude definition `Bounce`.
8. Change the element library to **Explicit**, and assign element type B31 to all regions of the model.
9. Create a new job named `expDynCrane`, and submit it for analysis.

When the job completes, enter the Visualization module to examine the results. In particular, compare the tip displacement history obtained earlier from Abaqus/Standard with that obtained from Abaqus/Explicit. As shown in *Figure 2*, there are small differences in the response. These differences are due to the different element and damping types used for the modal dynamic analysis. In fact, if the Abaqus/Standard analysis is modified to use B31 elements and mass proportional damping, the results produced by the two analysis products are nearly indistinguishable (see *Figure 2*), which confirms the accuracy of the modal dynamic procedure.

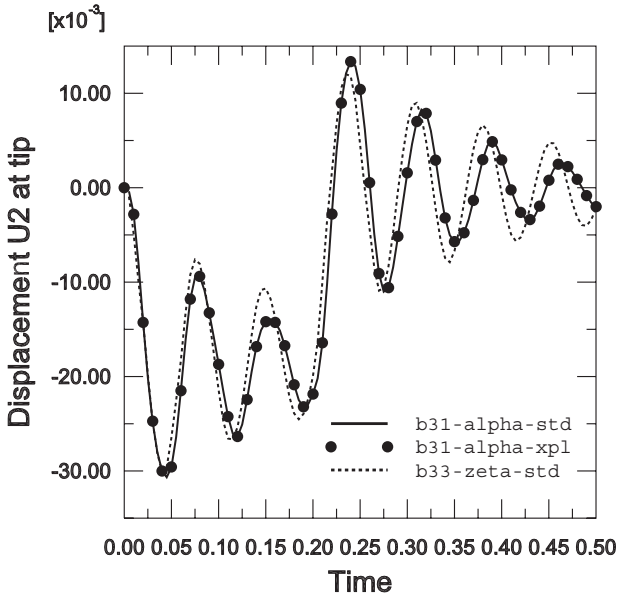


Figure 2: Comparison of tip displacements obtained from Abaqus/Standard and Abaqus/Explicit.

Other dynamic procedures

We now briefly review the other dynamic procedures available in Abaqus—namely linear modal dynamics and nonlinear dynamics.

In this section:

- *Linear modal dynamics*
- *Nonlinear dynamics*

Linear modal dynamics

There are several other linear, dynamic procedures in Abaqus/Standard that employ the modal superposition technique. Unlike the modal dynamics procedure, which calculates the response in the time domain, these procedures provide results in the frequency domain, which can give additional insight into the behavior of the structure.

A complete description of these procedures is given in *Dynamic Stress/Displacement Analysis*.

Steady-state dynamics

This procedure calculates the amplitude and phase of the structure's response caused by harmonic excitation over a user-specified range of frequencies. Typical examples include the following:

- The response of car engine mounts over a range of engine operating speeds.
- Rotating machinery in buildings.
- Components on aircraft engines.

Response spectrum

This procedure provides an estimate of the peak response (displacement, stress, etc.) when a structure is subjected to dynamic motion of its fixed points. The motion of the fixed points is known as “base motion”; an example is a seismic event causing ground motion. Typically the method is used when an estimate of the peak response is required for design purposes.

Random response

This procedure predicts the response of a system subjected to random continuous excitation. The excitation is expressed in a statistical sense using a power spectral density function. Examples of random response analysis include the following:

- The response of an airplane to turbulence.
- The response of a structure to noise, such as that emitted by a jet engine.

Nonlinear dynamics

As mentioned earlier, the modal dynamics procedure is suitable only for linear problems. When nonlinear dynamic response is of interest, the equations of motion must be integrated directly. The direct-integration of the equations of motion is performed in Abaqus/Standard using an implicit dynamics procedure. When this procedure is used, the mass, damping, and stiffness matrices are assembled and the equation of dynamic equilibrium is solved at each point in time. Since these operations are computationally intensive, direct-integration dynamics is more expensive than the modal methods.

Since the nonlinear dynamic procedure in Abaqus/Standard uses implicit time integration, it is suitable for nonlinear structural dynamics problems, for example, in which a sudden event initiates the dynamic response, such as an impact, or when the structural response involves large amounts of energy being dissipated by plasticity or viscous damping. In such studies the high frequency response, which is important initially, is damped out rapidly by the dissipative mechanisms in the model.

An alternative for nonlinear dynamic analyses is the explicit dynamics procedure available in Abaqus/Explicit. As discussed in *Abaqus Basics*, the explicit algorithm propagates the solution as a stress wave through the model, one element at a time. Thus, it is most suitable for applications in which stress wave effects are important and in which the event time being simulated is short (typically less than one second).

Another advantage associated with the explicit algorithm is that it can model discontinuous nonlinearities such as contact and failure more easily than Abaqus/Standard. Large, highly discontinuous problems are often more easily modeled with Abaqus/Explicit, even if the response is quasi-static. Explicit dynamic analyses are discussed further in *Nonlinear Explicit Dynamics*.

Related Abaqus examples

Related Abaqus examples are available.

- *Linear analysis of the Indian Point reactor feedwater line*
- *Explosively loaded cylindrical panel*
- *Eigenvalue analysis of a cantilever plate*

Suggested reading

This section lists texts and papers that are recommended reading for obtaining additional information.

- Clough, R. W. and J. Penzien, *Dynamics of Structures*, McGraw-Hill, 1975.
- NAFEMS Ltd., *A Finite Element Dynamics Primer*, 1993.
- Spence, P. W. and C. J. Kenchington, *The Role of Damping in Finite Element Analysis*, Report R0021, NAFEMS Ltd., 1993.

Summary

This section presents a summary of the concepts.

- Dynamic analyses include the effect of the structure's inertia.
- The frequency extraction procedure in Abaqus/Standard extracts the natural frequencies and mode shapes of the structure.
- The mode shapes can then be used to determine the dynamic response of linear systems by modal superposition. This technique is efficient, but it cannot be used for nonlinear problems.
- Linear dynamic procedures are available in Abaqus/Standard to calculate the transient response to transient loading, the steady-state response to harmonic loading, the peak response to base motion, and the response to random loading.
- You should extract enough modes to obtain an accurate representation of the dynamic behavior of the structure. The total modal effective mass in the direction in which motion will occur should be at least 90% of the mass that can move to produce accurate results.
- You can define direct modal damping, Rayleigh damping, and composite modal damping in Abaqus/Standard. However, since the natural frequencies and mode shapes are based on the undamped structure, the structure being analyzed should be only lightly damped.
- Modal techniques are not suitable for nonlinear dynamic simulations. Direct time integration methods or explicit analysis must be used in these situations.
- Time variation of loads or prescribed boundary conditions can be defined with an amplitude curve.
- Mode shapes and transient results can be animated in the Visualization module of Abaqus/CAE. This provides a useful way of understanding the response of dynamic and nonlinear static analyses.

Nonlinearity

This chapter discusses nonlinear structural analysis in Abaqus.

In this section:

- *Linear analyses versus nonlinear analyses*
- *Sources of nonlinearity*
- *The solution of nonlinear problems*
- *Including nonlinearity in an Abaqus analysis*
- *Example: nonlinear skew plate*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Linear analyses versus nonlinear analyses

The differences between linear and nonlinear analyses are summarized below.

Linear analysis

All the analyses discussed so far have been linear: there is a linear relationship between the applied loads and the response of the system. For example, if a linear spring extends statically by 1 m under a load of 10 N, it will extend by 2 m when a load of 20 N is applied. This means that in a linear Abaqus/Standard analysis the flexibility of the structure need only be calculated once (by assembling the stiffness matrix and inverting it). The linear response of the structure to other load cases can be found by multiplying the new vector of loads by the inverted stiffness matrix. Furthermore, the structure's response to various load cases can be scaled by constants and/or superimposed on one another to determine its response to a completely new load case, provided that the new load case is the sum (or multiple) of previous ones. This principle of superposition of load cases assumes that the same boundary conditions are used for all the load cases.

Abaqus/Standard uses the principle of superposition of load cases in linear dynamics simulations, which are discussed in *Linear Dynamics*.

Nonlinear analysis

A nonlinear structural problem is one in which the structure's stiffness changes as it deforms. All physical structures exhibit nonlinear behavior. Linear analysis is a convenient approximation that is often adequate for design purposes. It is obviously inadequate for many structural simulations including manufacturing processes, such as forging or stamping; crash analyses; and analyses of rubber components, such as tires or engine mounts. A simple example is a spring with a nonlinear stiffening response (see *Figure 1*).

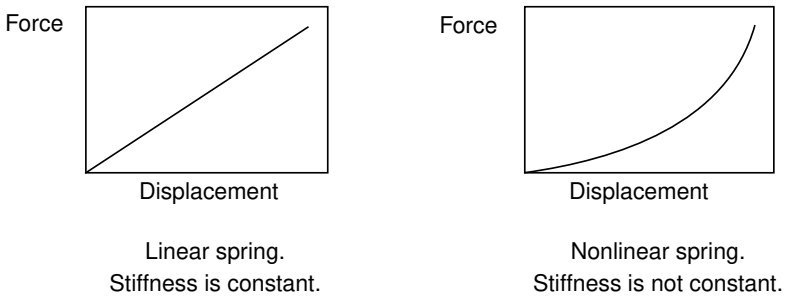


Figure 1: Linear and nonlinear spring characteristics.

Since the stiffness is now dependent on the displacement, the initial flexibility can no longer be multiplied by the applied load to calculate the spring's displacement for any load. In a nonlinear implicit analysis the stiffness matrix of the structure has to be assembled and inverted many times during the course of the analysis, making it much more expensive to solve than a linear implicit analysis. In an explicit analysis the increased cost of a nonlinear analysis is due to reductions in the stable time increment. The stable time increment is discussed further in *Nonlinear Explicit Dynamics*.

Since the response of a nonlinear system is not a linear function of the magnitude of the applied load, it is not possible to create solutions for different load cases by superposition. Each load case must be defined and solved as a separate analysis.

Sources of nonlinearity

The sources of nonlinearity in structural mechanics simulations are material nonlinearity, boundary nonlinearity, and geometric nonlinearity.

In this section:

- *Material nonlinearity*
- *Boundary nonlinearity*
- *Geometric nonlinearity*

Material nonlinearity

This type of nonlinearity is probably the one that you are most familiar with and is covered in more depth in *Materials*. Most metals have a fairly linear stress–strain relationship at low strain values; but at higher strains the material yields, at which point the response becomes nonlinear and irreversible (see *Figure 1*).

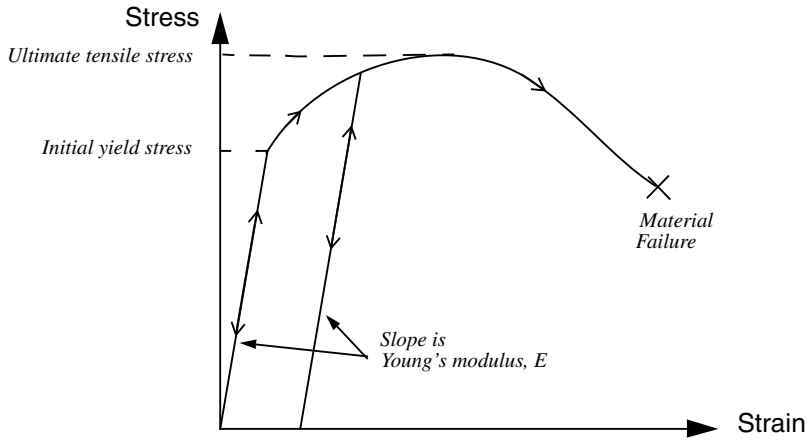


Figure 1: Stress-strain curve for an elastic-plastic material under uniaxial tension.

Rubber materials can be approximated by a nonlinear, reversible (elastic) response (see *Figure 2*).

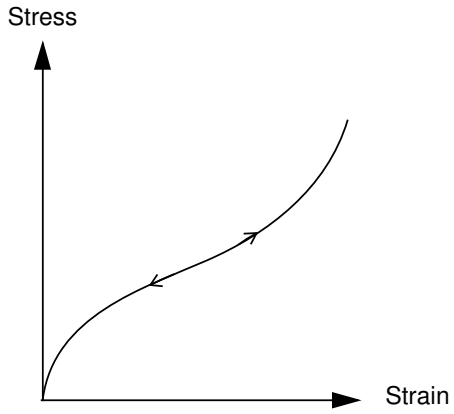


Figure 2: Stress-strain curve for a rubber-type material.

Material nonlinearity may be related to factors other than strain. Strain-rate-dependent material data and material failure are both forms of material nonlinearity. Material properties can also be a function of temperature and other predefined fields.

Boundary nonlinearity

Boundary nonlinearity occurs if the boundary conditions change during the analysis. Consider the cantilever beam, shown in *Figure 1*, that deflects under an applied load until it hits a "stop."

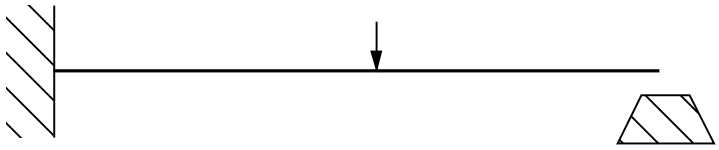


Figure 1: Cantilever beam hitting a stop.

The vertical deflection of the tip is linearly related to the load (if the deflection is small) until it contacts the stop. There is then a sudden change in the boundary condition at the tip of the beam, preventing any further vertical deflection, and so the response of the beam is no longer linear. Boundary nonlinearities are extremely discontinuous: when contact occurs during a simulation, there is a large and instantaneous change in the response of the structure.

Another example of boundary nonlinearity is blowing a sheet of material into a mold. The sheet expands relatively easily under the applied pressure until it begins to contact the mold. From then on the pressure must be increased to continue forming the sheet because of the change in boundary conditions.

Boundary nonlinearity is covered in *Contact*.

Geometric nonlinearity

The third source of nonlinearity is related to changes in the geometry of the structure during the analysis. Geometric nonlinearity occurs whenever the magnitude of the displacements affects the response of the structure. This may be caused by:

- Large deflections or rotations.
- “Snap through.”
- Initial stresses or load stiffening.

For example, consider a cantilever beam loaded vertically at the tip (see *Figure 1*).

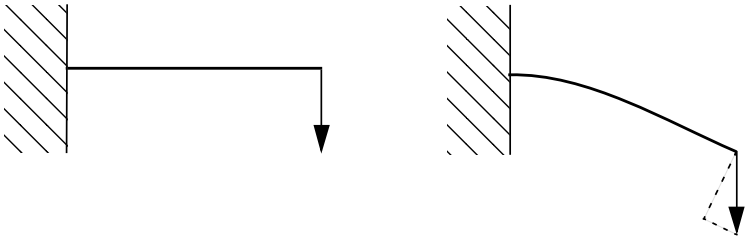


Figure 1: Large deflection of a cantilever beam.

If the tip deflection is small, the analysis can be considered as being approximately linear. However, if the tip deflections are large, the shape of the structure and, hence, its stiffness changes. In addition, if the load does not remain perpendicular to the beam, the action of the load on the structure changes significantly. As the cantilever beam deflects, the load can be resolved into a component perpendicular to the beam and a component acting along the length of the beam. Both of these effects contribute to the nonlinear response of the cantilever beam (i.e., the changing of the beam's stiffness as the load it carries increases).

One would expect large deflections and rotations to have a significant effect on the way that structures carry loads. However, displacements do not necessarily have to be large relative to the dimensions of the structure for geometric nonlinearity to be important. Consider the “snap through” under applied pressure of a large panel with a shallow curve, as shown in *Figure 2*.

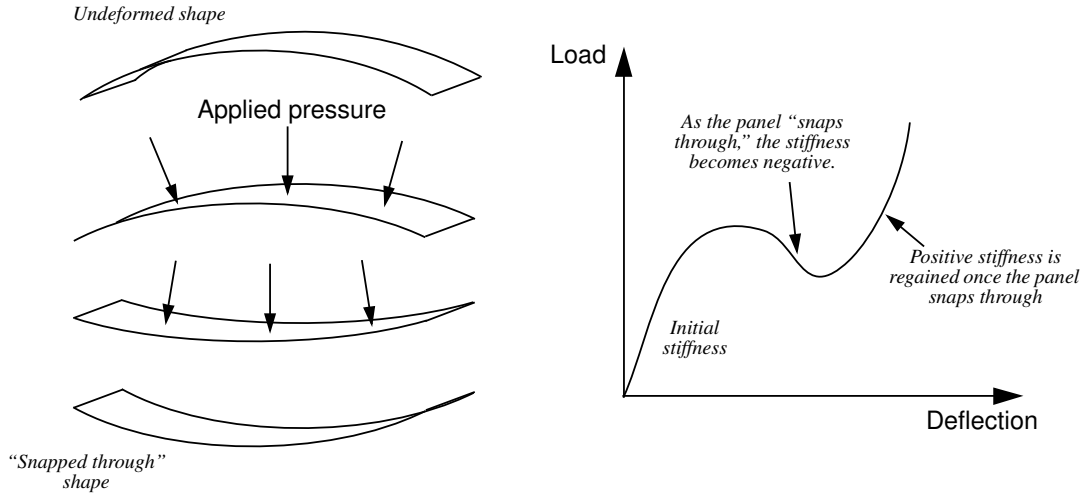


Figure 2: Snap-through of a large shallow panel.

In this example there is a dramatic change in the stiffness of the panel as it deforms. As the panel “snaps through,” the stiffness becomes negative. Thus, although the magnitude of the displacements, relative to the panel's dimensions, is quite small, there is significant geometric nonlinearity in the simulation, which must be taken into consideration.

An important difference between the analysis products should be noted here: by default, Abaqus/Standard assumes small deformations, while Abaqus/Explicit assumes large deformations.

The solution of nonlinear problems

This section discusses how Abaqus solves nonlinear problems.

In this section:

- *Nonlinear load-displacement curve*
- *Steps, increments, and iterations*
- *Equilibrium iterations and convergence in Abaqus/Standard*
- *Automatic incrementation control in Abaqus/Standard*

Nonlinear load-displacement curve

The nonlinear load-displacement curve for a structure is analyzed.

The nonlinear load-displacement curve for a structure is shown in *Figure 1*.

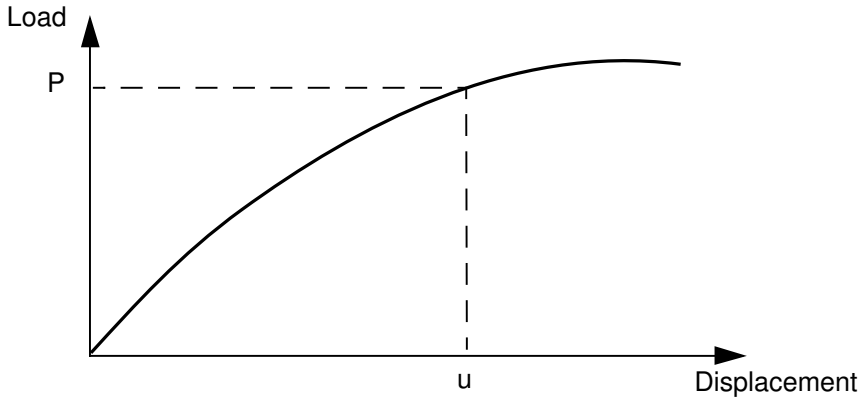
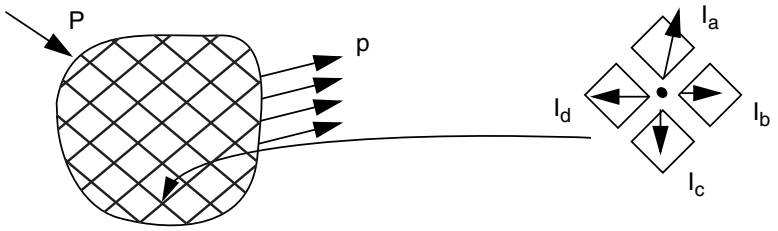


Figure 1: Nonlinear load-displacement curve.

The objective of the analysis is to determine this response. Consider the external forces, P, and the internal (nodal) forces, I, acting on a body (see *Figure 2*(a) and *Figure 2*(b), respectively).



(a) External loads in a simulation. (b) Internal forces acting at a node.

Figure 2: Internal and external loads on a body.

The internal loads acting on a node are caused by the stresses in the elements that contain that node.

For the body to be in static equilibrium, the net force acting at every node must be zero. Therefore, the basic statement of static equilibrium is that the internal forces, I, and the external forces, P, must balance each other:

$$\boldsymbol{P - I = 0.}$$

Abaqus/Standard uses the Newton-Raphson method to obtain solutions for nonlinear problems. In a nonlinear analysis the solution usually cannot be calculated by solving a single system of equations, as would be done in a linear problem. Instead, the solution is found by applying the specified loads gradually and incrementally working toward the final solution. Therefore, Abaqus/Standard breaks the simulation into a number of *load increments* and finds the approximate equilibrium configuration at the end of each load increment. It often takes Abaqus/Standard several iterations to determine an acceptable solution to a given load increment. The sum of all of the incremental responses is the approximate solution for the nonlinear analysis. Thus, Abaqus/Standard combines incremental and iterative procedures for solving nonlinear problems.

Abaqus/Explicit determines a solution to the dynamic equilibrium equation $\boldsymbol{P - I = M\ddot{u}}$ without iterating by explicitly advancing the kinematic state from the end of the previous increment. Solving a problem explicitly does not require the formation of tangent stiffness matrices. 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 $\boldsymbol{t + \Delta t/2}$ and the displacement solution to time $\boldsymbol{t + \Delta t}$. For linear and nonlinear problems alike, explicit methods require a small time increment size that depends solely on the highest natural frequency of the model and is independent of the type and duration of loading. Simulations typically require a large number of increments; however, due to the fact that a global set of equations is not solved in each increment, the cost per increment of an explicit method is much smaller than that of an implicit method. The small increments characteristic of an explicit dynamic method make Abaqus/Explicit well suited for nonlinear analysis.

Steps, increments, and iterations

This section introduces some new vocabulary for describing the various parts of an analysis. It is important that you clearly understand the difference between an analysis *step*, a load *increment*, and an *iteration*.

- The load history for a simulation consists of one or more steps. You define the steps, which generally consist of an analysis procedure option, loading options, and output request options. Different loads, boundary conditions, analysis procedure options, 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 the total load applied in a step is broken into smaller increments so that the nonlinear solution path can be followed.

  In Abaqus/Standard you suggest the size of the first increment, and Abaqus/Standard automatically chooses the size of the subsequent increments. In Abaqus/Explicit the default time incrementation is fully automatic and does not require user intervention. Because the explicit method is conditionally stable, there is a stability limit for the time increment. The stable time increment is discussed in *Nonlinear Explicit Dynamics*.

  At the end of each increment the structure is in (approximate) equilibrium and results are available for writing to the output database, restart, data, or results files. The increments at which you select results to be written to the output database file are called *frames*.

  The issues associated with time incrementation in Abaqus/Standard and Abaqus/Explicit analyses are quite different, since time increments are generally much smaller in Abaqus/Explicit.
- An iteration is an attempt at finding an equilibrium solution in an increment when solving with an implicit method. If the model is not in equilibrium at the end of the iteration, Abaqus/Standard tries another iteration. With every iteration the solution Abaqus/Standard obtains should be closer to equilibrium; sometimes Abaqus/Standard may need many iterations to obtain an equilibrium solution. When an equilibrium solution has been obtained, the increment is complete. Results can be requested only at the end of an increment.

  Abaqus/Explicit does not need to iterate to obtain the solution in an increment.

Equilibrium iterations and convergence in Abaqus/Standard

The nonlinear response of a structure to a small load increment, ΔP, is shown in *Figure 1*. Abaqus/Standard uses the structure's initial stiffness, K_0, which is based on its configuration at u_0, and ΔP to calculate a *displacement correction*, c_a, for the structure. Using c_a, the structure's configuration is updated to u_a.

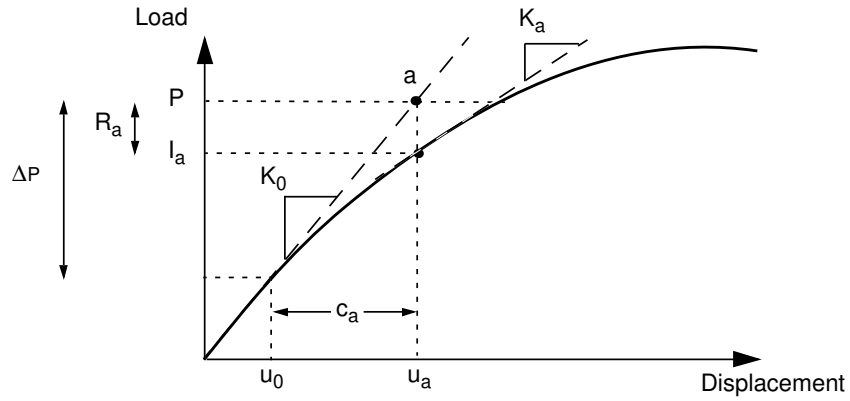


Figure 1: First iteration in an increment.

Convergence

Abaqus/Standard forms a new stiffness, K_a, for the structure, based on its updated configuration, u_a. Abaqus/Standard also calculates I_a, in this updated configuration. The difference between the total applied load, P, and I_a can now be calculated as:

$$R_a = P - I_a,$$

where R_a is the *force residual* for the iteration.

If R_a is zero at every degree of freedom in the model, point *a* in *Figure 1* would lie on the load-deflection curve, and the structure would be in equilibrium. In a nonlinear problem it is almost impossible to have R_a equal zero, so Abaqus/Standard compares it to a tolerance value. If R_a is less than this force residual tolerance, Abaqus/Standard accepts the structure's updated configuration as the equilibrium solution. 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.

If R_a is less than the current tolerance value, P and I_a are in equilibrium, and 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 displacement correction, c_a, is small relative to the total incremental displacement, $\Delta u_a = u_a - u_0$. If c_a is greater than 1% 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 load increment. The exception to this rule is the case of a *linear* increment, which is defined as any increment in which the largest force residual is less than 10^{-8} times the time-averaged force. Any case that passes such a stringent comparison of the largest force residual with the time-averaged force is considered linear and does not require further iteration. The solution is accepted without any check on the size of the displacement correction.

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. This second iteration uses the stiffness, K_a, calculated at the end of

the previous iteration together with $\boldsymbol{R_a}$ to determine another displacement correction, $\boldsymbol{c_b}$, that brings the system closer to equilibrium (point *b* in *Figure 2*).

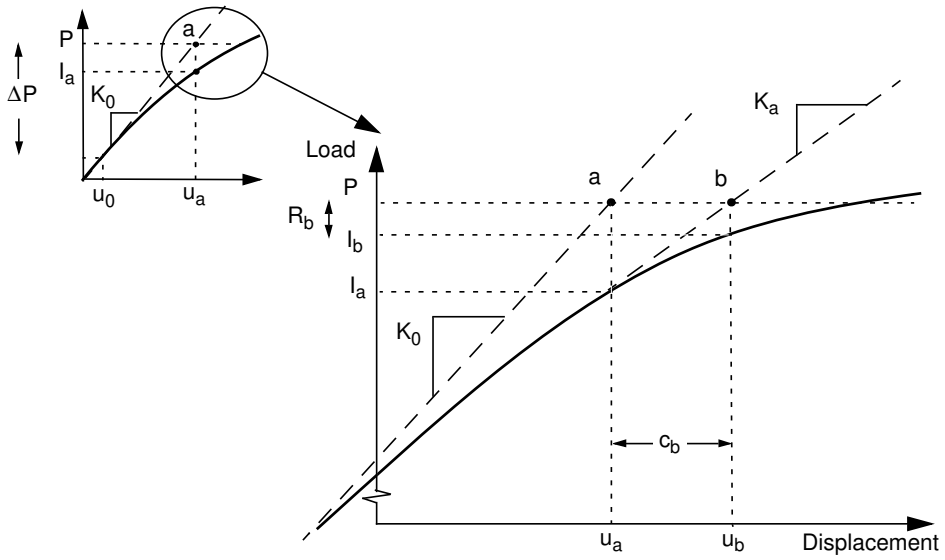


Figure 2: Second iteration.

Abaqus/Standard calculates a new force residual, $\boldsymbol{R_b}$, using the internal forces from the structure's new configuration, $\boldsymbol{u_b}$. Again, the largest force residual at any degree of freedom, $\boldsymbol{R_b}$, is compared against the force residual tolerance, and the displacement correction for the second iteration, $\boldsymbol{c_b}$, is compared to the increment of displacement, $\boldsymbol{\Delta u_b = u_b - u_0}$. If necessary, Abaqus/Standard performs further iterations.

For each iteration in a nonlinear analysis Abaqus/Standard forms the model's stiffness matrix and solves a system of equations. This means that each iteration is equivalent, in computational cost, to conducting a complete linear analysis. It should now be clear that the computational expense of a nonlinear analysis in Abaqus/Standard can be many times greater than for a linear one.

It is possible with Abaqus/Standard to save results at each converged increment. Thus, the amount of output data available from a nonlinear simulation is many times that available from a linear analysis of the same geometry. Consider both of these factors and the types of nonlinear simulations that you want to perform when planning your computer resources.

Automatic incrementation control in Abaqus/Standard

Abaqus/Standard automatically adjusts the size of the load increments so that it solves nonlinear problems easily and efficiently. You only need to suggest the size of the first increment in each step of your simulation. Thereafter, Abaqus/Standard automatically adjusts the size of the increments. If you do not provide a suggested initial increment size, Abaqus/Standard will try to apply all of the loads defined in the step in the first increment. In highly nonlinear problems Abaqus/Standard will have to reduce the increment size repeatedly, resulting in wasted CPU time. Generally it is to your advantage to provide a reasonable initial increment size (see *Modifications to the model* for an example); only in very 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 load increment will vary depending on the degree of nonlinearity in the system. By default, 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. An attempt is then made at finding a converged solution with this smaller load increment. If the increment still fails to converge, Abaqus/Standard reduces the increment size again. By default, Abaqus/Standard allows a maximum of five cutbacks of increment size in an increment before stopping the analysis.

In Abaqus/Standard you can also specify the maximum number of increments allowed during the step. Abaqus/Standard terminates the analysis with an error message if it needs more increments than this limit to complete the step. The default number of increments for a step is 100; if significant nonlinearity is present in the simulation, the analysis may require many more increments. You specify an upper limit on the number of increments that Abaqus/Standard can use, rather than the number of increments it must use.

In a nonlinear analysis a step takes place over a finite period of "time," although this "time" has no physical meaning unless inertial effects or rate-dependent behavior are present. In Abaqus/Standard you specify the initial time increment, $\Delta T_{initial}$, and the total step time, T_{total}. The ratio of the initial time increment to the step time specifies the proportion of load applied in the first increment. The initial load increment is given by

$$\frac{\Delta T_{initial}}{T_{total}} \times \text{Load Magnitude.}$$

The choice of initial time increment can be critical in certain nonlinear simulations in Abaqus/Standard, but for most analyses an initial increment size that is 5% to 10% of the total step time is usually sufficient. In static simulations the total step time is usually set to 1.0 for convenience, unless, for example, rate-dependent material effects or dashpots are included in the model. With a total step time of 1.0 the proportion of load applied is always equal to the current step time; i.e., 50% of the total load is applied when the step time is 0.5.

Although you must specify the initial increment size in Abaqus/Standard, Abaqus/Standard automatically controls the size of the subsequent increments. This automatic control of the increment size is suitable for the majority of nonlinear simulations performed with Abaqus/Standard, although further controls on the increment size are available. Abaqus/Standard will terminate an analysis if excessive cutbacks caused by convergence problems reduce the increment size below the minimum value. The default minimum allowable time increment, ΔT_{min}, is 10^{-5} times the total step time. By default, Abaqus/Standard has no upper limit on the increment size, ΔT_{max}, other than the total step time. Depending on your Abaqus/Standard simulation, you may want to specify different minimum and/or maximum allowable increment sizes. For example, if you know that your simulation may have trouble obtaining a solution if too large a load increment is applied, perhaps because the model may undergo plastic deformation, you may want to decrease ΔT_{max}.

If the increment converges in fewer than five iterations, this indicates that the solution is being found fairly easily. Therefore, Abaqus/Standard automatically increases the increment size by 50% if two consecutive increments require fewer than five iterations to obtain a converged solution.

Details of the automatic load incrementation scheme are given in the **Job Diagnostics** dialog box, as shown in more detail in *Job diagnostics*.

Including nonlinearity in an Abaqus analysis

We now discuss how to account for nonlinearity in an Abaqus analysis.
The main focus is on geometric nonlinearity.

Geometric nonlinearity

Incorporating the effects of geometric nonlinearity in an analysis requires only minor changes to an Abaqus/Standard model. You need to make sure the step definition considers geometrically nonlinear effects. This is the default setting in Abaqus/Explicit. You also need to set time incrementation parameters as discussed in *Automatic incrementation control in Abaqus/Standard*.

Local directions

In a geometrically nonlinear analysis the local material directions may rotate with the deformation in each element. For shell, beam, and truss elements the local material directions always rotate with the deformation. For solid elements the local material directions rotate with the deformation only if the elements refer to nondefault local material directions; otherwise, the default local material directions remain constant throughout the analysis.

Local directions defined at nodes remain fixed throughout the analysis; they do not rotate with the deformation. See *Transformed Coordinate Systems* for further details.

Effect on subsequent steps

Once you include geometric nonlinearity in a step, it is considered in all subsequent steps. If nonlinear geometric effects are not requested in a subsequent step, Abaqus will issue a warning stating that they are being included in the step anyway.

Other geometrically nonlinear effects

The large deformations in a model are not the only important effects that are considered when geometric nonlinearity is activated. Abaqus/Standard also includes terms in the element stiffness calculations that are caused by the applied loads, the so-called load stiffness. These terms improve convergence behavior. In addition, the membrane loads in shells and the axial loads in cables and beams contribute much of the stiffness of these structures in response to transverse loads. By including geometric nonlinearity, the membrane stiffness in response to transverse loads is considered as well.

Material nonlinearity

The addition of material nonlinearity to an Abaqus model is discussed in *Materials*.

Boundary nonlinearity

The introduction of boundary nonlinearity is discussed in *Contact*.

Example: nonlinear skew plate

This example models a nonlinear skew plate.

In this section:

- *Modifications to the model*
- *Job diagnostics*
- *Postprocessing*
- *Running the analysis in Abaqus/Explicit*

Modifications to the model

This example is a continuation of the linear skew plate simulation.
The example is described in *Using Shell Elements* and shown in *Figure 1*.

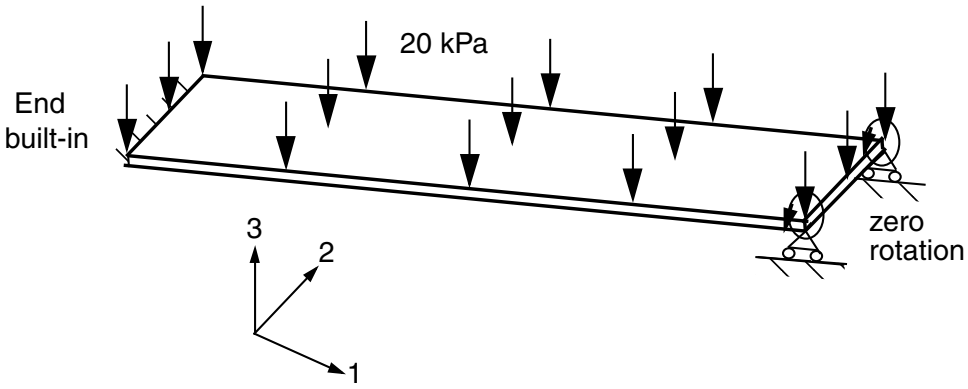


Figure 1: Skew plate.

You will now reanalyze the plate in Abaqus/Standard to include the effects of geometric nonlinearity. The results from this analysis will allow you to determine the importance of geometrically nonlinear effects and, therefore, the validity of the linear analysis.

If you wish, you can follow the guidelines at the end of this example to extend the simulation to perform a dynamic analysis using Abaqus/Explicit.

Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Nonlinear skew plate*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Nonlinear skew plate**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Open the model database file `SkewPlate.cae`. Copy the model named `Linear` to a model named `Nonlinear`.

For the `Nonlinear` skew plate model, you will include nonlinear geometric effects as well as change the output requests.

Defining the step

In the Model Tree, double-click the **Apply Pressure** step underneath the **Steps** container to edit the step definition. In the **Basic** tabbed page of the **Edit Step** dialog box, toggle on **Nlgeom** to include geometric nonlinearity effects, and ensure the time period for the step is set to `1.0`. In the **Incrementation** tabbed page, set the initial increment size to `0.1`. The default maximum number of increments is `100`; Abaqus may use fewer increments than this upper limit, but it will stop the analysis if it needs more.

You may wish to change the description of the step to reflect that it is now a nonlinear analysis step.

Output control

In a linear analysis Abaqus solves the equilibrium equations once and calculates the results for this one solution. A nonlinear analysis can produce much more output because results can be requested at the end of each converged

increment. If you do not select the output requests carefully, the output files become very large, potentially filling the disk space on your computer.

As noted earlier, output is available in four different files:

- the output database (`.odb`) file, which contains data in a neutral binary format necessary to postprocess the results with Abaqus/CAE;
- the data (`.dat`) file, which contains printed tables of selected results (available only in Abaqus/Standard);
- the restart (`.res`) file, which is used to continue the analysis; and
- the results (`.fil`) file, which is used with third-party postprocessors.

Only the output database (`.odb`) file is discussed here. If selected carefully, data can be saved frequently during the simulation without using excessive disk space.

Open the **Field Output Requests Manager**. On the right side of the dialog box, click **Edit** to open the field output editor. Remove the field output requests defined for the linear analysis model, and specify the default field output requests by selecting **Preselected defaults** under **Output Variables**. This preselected set of output variables is the most commonly used set of field variables for a general static procedure.

To reduce the size of the output database file, write field output every second increment. If you were simply interested in the final results, you could either select **Last increment** or set the frequency at which output is saved equal to a large number. Results are always stored at the end of each step, regardless of the value specified; therefore, using a large value causes only the final results to be saved.

The history output request for the displacements of the nodes at the midspan can be kept from the previous analysis. You will explore these results using the *X–Y* plotting capability in the Visualization module.

Running and monitoring the job

Create a job for the `Nonlinear` model named `NlSkewPlate`, and give it the description `Nonlinear Elastic Skew Plate`. Remember to save your model in a new model database file.

Submit the job for analysis, and monitor the solution progress. If any errors are encountered, correct them; if any warning messages are issued, investigate their source and take corrective action as necessary.

Figure 2 shows the contents of the **Job Monitor** for this nonlinear skew plate simulation.

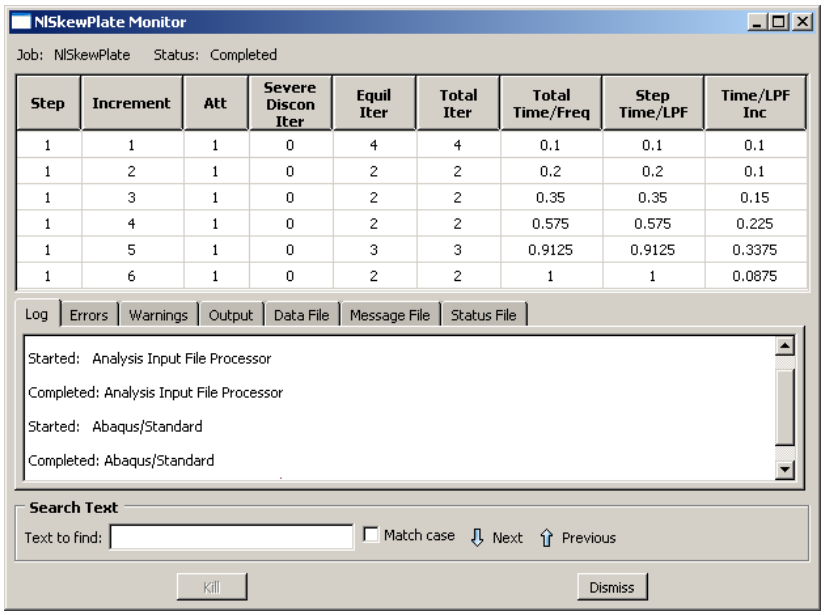


Figure 2: Job Monitor: nonlinear skew plate analysis.

The first column shows the step number—in this case there is only one step. The second column gives the increment number. The sixth column shows the number of iterations Abaqus/Standard needed to obtain a converged solution in each increment; for example, Abaqus/Standard needed four iterations in increment 1. The eighth column shows the total step time completed, and the ninth column shows the increment size (ΔT).

This example shows how Abaqus/Standard automatically controls the increment size and, therefore, the proportion of load applied in each increment. In this analysis Abaqus/Standard applied 10% of the total load in the first increment: you specified $\Delta T_{initial}$ to be 0.1 and the step time to be 1.0. Abaqus/Standard needed four iterations to converge to a solution in the first increment. Abaqus/Standard only needed two iterations in the second increment, so it automatically increased the size of the next increment by 50% to $\Delta T = 0.15$. Abaqus/Standard also increased ΔT in both the fourth and fifth increments. It adjusted the final increment size to be just enough to complete the analysis; in this case the final increment size was 0.0875.

Job diagnostics

In addition to allowing you to monitor the progress of your analysis job, Abaqus/CAE provides a visual diagnostics tool to help you understand the convergence behavior of your job and debug the model if necessary. Abaqus/Standard stores information in the output database for each step, increment, attempt, and iteration of an analysis. This diagnostic information is saved automatically for every job that you run. If an analysis takes longer than expected or terminates prematurely, you can view the job diagnostic information in Abaqus/CAE to help determine the cause and to identify ways to correct the model.

Enter the Visualization module, and open the output database file `NlSkewPlate.odb` to examine the convergence history. From the main menu bar, select **Tools->Job Diagnostics** to open the **Job Diagnostics** dialog box. Click the "+" symbols in the **Job History** list to expand the list to include the steps, increments, attempts, and iterations in the analysis job. For example, under **Increment 1**, select **Attempt 1**, as shown in *Figure 1*.

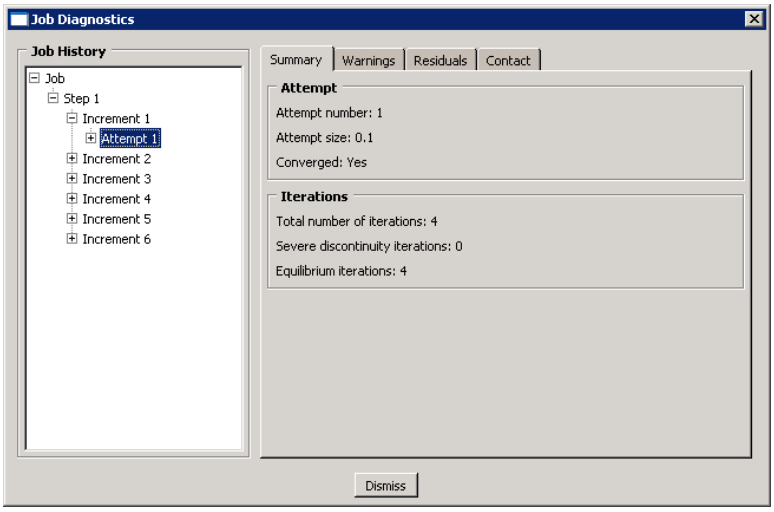


Figure 1: Summary information for the first attempt of the first increment.

The **Attempt** information on the right side of the dialog box contains basic information such as the increment size; the **Iterations** information contains the number of iterations attempted. Select **Iteration 1** of this attempt to see detailed information regarding the first iteration. The information on the **Summary** tabbed page tells you that convergence was not achieved in this iteration, so click the **Residuals** tab to understand why.

As shown in *Figure 2*, the **Residuals** tabbed page displays the values of the average force, $\boldsymbol{q}^{\alpha}$, and time average force, $\tilde{\boldsymbol{q}}^{\alpha}$, in the model.

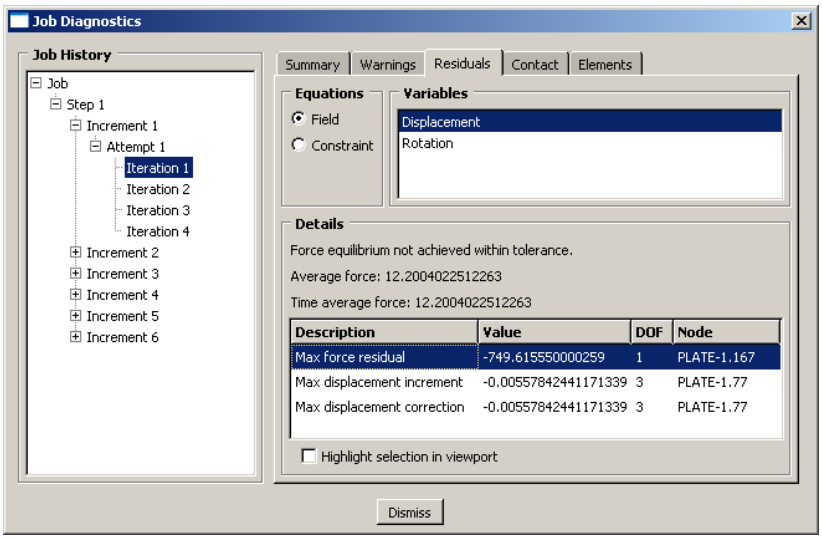


Figure 2: Force residual information for the first iteration.

It also displays the values of the largest residual force, r^{α}_{max}, the largest increment of displacement, Δu_{α}, and the largest correction to displacement, c_{α}, as well as the nodes and degrees of freedom (DOF) at which these occur. The location in the model where any of these occur can be highlighted in the viewport by toggling on **Highlight selection in viewport** at the bottom of the dialog box. The selection of the diagnostic criterion is persistent. Thus, you can click your way through the list of iterations on the left side of the dialog box to quickly see how the location associated with a given criterion changes throughout the iteration history. This may prove very useful if you are trying to debug a large, complex model. A similar display is available for rotational degrees of freedom (select **Rotation** under the list of **Variables**).

In this example the initial time increment is 0.1, as specified in the step definition. The average force for the increment is 12.2 N; and the time average force, $\tilde{q}^{\alpha}$, has the same value since this is the first increment. The largest residual force in the model, r^{α}_{max}, is −749.6 N, which is clearly greater than $0.005 \times \tilde{q}^{\alpha}$. r^{α}_{max} occurred at node 167 in degree of freedom 1 (your values and node numbers may vary). Abaqus/Standard must also check for equilibrium of the moments in the model since this model includes shell elements. The moment/rotation field also failed to satisfy the equilibrium check.

Although failure to satisfy the equilibrium check is enough to cause Abaqus/Standard to try another iteration, you should also examine the displacement correction. In the first iteration of the first increment of the first step the largest increment of displacement, Δu^{α}_{max}, and the largest correction to displacement, c^{α}_{max}, are both -5.578×10^{-3} m; and the largest increment of rotation and correction to rotation are both -1.598×10^{-2} radians. Since the incremental values and the corrections are always equal in the first iteration of the first increment of the first step, the check that the largest corrections to nodal variables are less than 1% of the largest incremental values will always fail. However, if Abaqus/Standard judges the solution to be linear (a judgement based on the magnitude of the residuals, $r^{\alpha}_{max} < 10^{-8}\tilde{q}^{\alpha}$), it will ignore this criterion.

Since Abaqus/Standard did not find an equilibrium solution in the first iteration, it tries a second iteration. The residual information for the second iteration is shown in *Figure 3*.

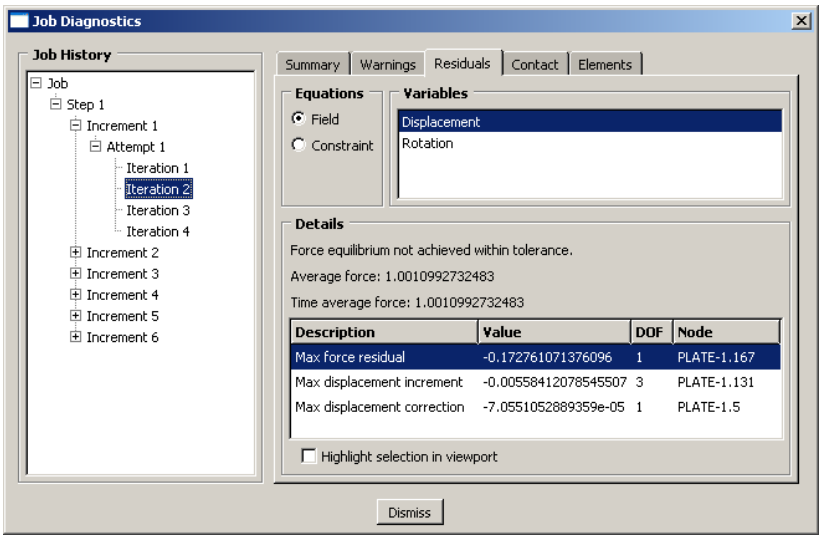


Figure 3: Force residual information for the second iteration.

In the second iteration r^{α}_{max} has fallen to −0.173 N at node 167 in degree of freedom 1. However, equilibrium is not satisfied in this iteration because $0.005 \times \tilde{q}^{\alpha}$, where $\tilde{q}^{\alpha}$ = 1.00 N, is still less than r^{α}_{max}. The displacement correction criterion also failed again because $c^{\alpha}_{max} = -7.055 \times 10^{-5}$, which occurred at node 5 in degree of freedom 1, is more than 1% of $\Delta u^{\alpha}_{max} = -5.584 \times 10^{-3}$, the maximum displacement increment.

Both the moment residual check and the largest correction to rotation check were satisfied in this second iteration; however, Abaqus/Standard must perform two more iterations because the solutions did not pass the force residual check (or the largest correction to displacement criterion). The residual information for the additional iterations necessary to obtain a solution in the first increment are shown in *Figure 4* and *Figure 5*.

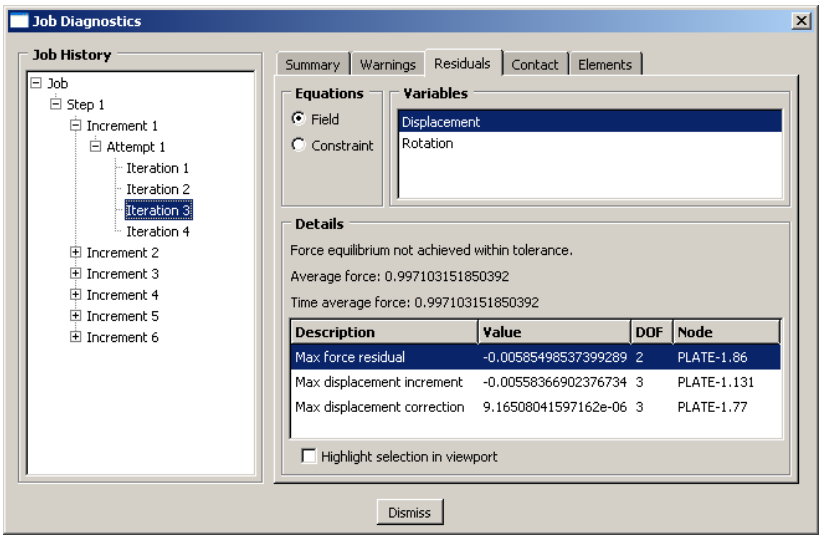


Figure 4: Force residual information for the third iteration.

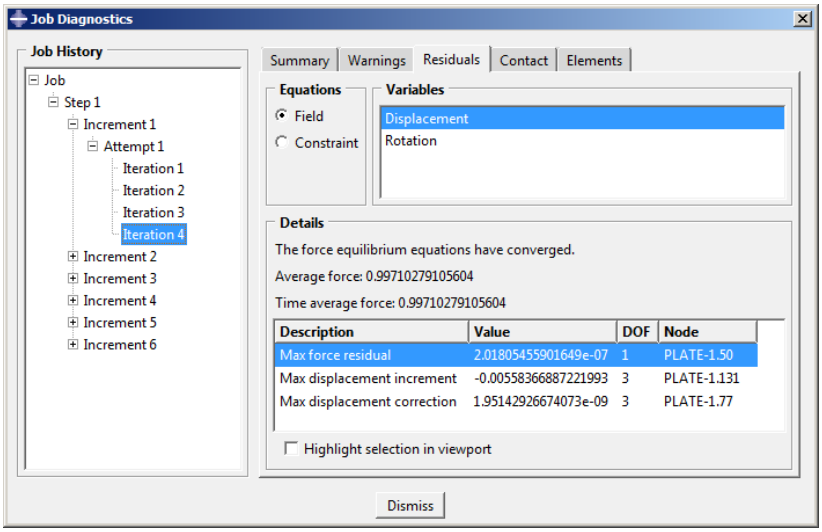


Figure 5: Force residual information for the fourth iteration.

After the fourth iteration $\tilde{q}^{\alpha} = 0.997$ N and $r^{\alpha}_{max} = 2.018 \times 10^{-7}$ N at node 50 in degree of freedom 1. These values satisfy $r^{\alpha}_{max} < 0.005 \times \tilde{q}^{\alpha}$, so the force residual check is satisfied. Comparing c^{α}_{max} to the largest increment of displacement shows that the displacement correction is below the required tolerance. The solution for the forces and displacements has, therefore, converged. The checks for both the moment residual and the rotation correction continue to be satisfied, as they have been since the second iteration. With a solution that satisfies equilibrium for all variables (displacement and rotation in this case), the first load increment is complete. The attempt summary (*Figure 1*) shows the number of iterations that were required for this increment and the size of the increment.

Abaqus/Standard continues this process of applying an increment of load then iterating to find a solution until it completes the whole analysis (or reaches the maximum increment that you specified). In this analysis it required five more increments. The step summary is shown in *Figure 6*.

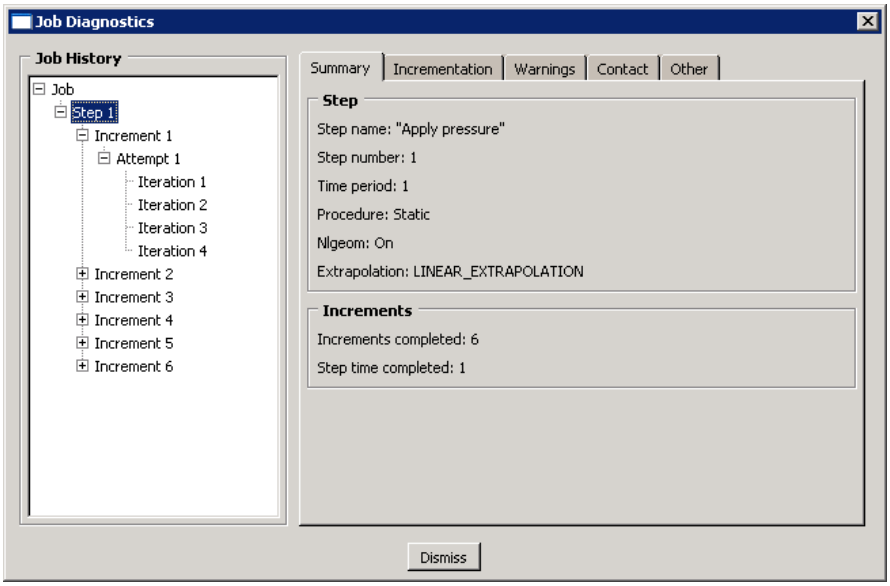


Figure 6: Analysis step summary.

In addition to the residual information discussed above, analysis warning messages related to numerical singularities, zero pivots, and negative eigenvalues, if they exist, are displayed in the **Job Diagnostics** dialog box (in the **Warnings** tabbed page). Always investigate the cause of these warnings.

Postprocessing

You will now postprocess the results.

In this section:

- *Showing the available frames*
- *Displaying the deformed and undeformed model shapes*
- *Using results from other frames*
- *X–Y plotting*
- *Tabular data*

Showing the available frames

To begin this exercise, determine the available output frames (the increment intervals at which results were written to the output database).

1. From the main menu bar, select **Result->Step/Frame**.

   The **Step/Frame** dialog box appears.

   During the analysis Abaqus/Standard wrote field output results to the output database file every second increment, as was requested. Abaqus/Viewer displays the list of the available frames, as shown in *Figure 1*.

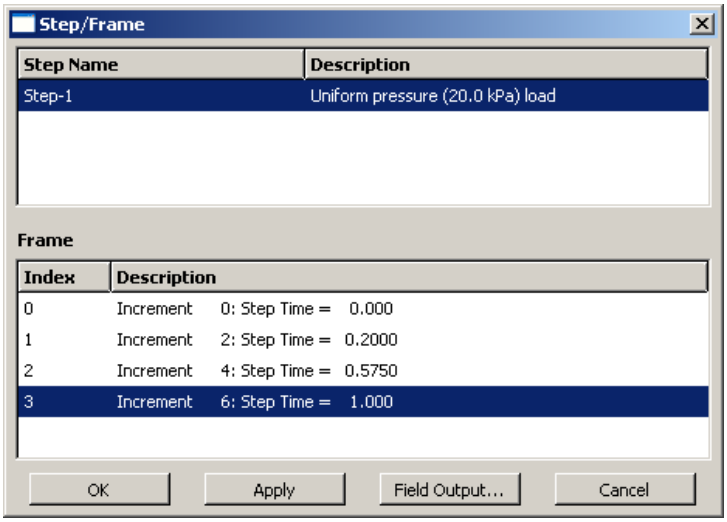


Figure 1: Available frames.

   The list tabulates the steps and increments for which field variables are stored. This analysis consisted of a single step with six increments. The results for increment 0 (the initial state of the step) are saved by default, and you saved data for increments 2, 4, and 6. By default, Abaqus/CAE always uses the data for the last available increment saved in the output database file.

2. Click **OK** to dismiss the **Step/Frame** dialog box.

Displaying the deformed and undeformed model shapes

You can display the deformed and undeformed model shapes with the **Allow Multiple Plot States** tool.

Use the **Allow Multiple Plot States**  tool to display the deformed model shape with the undeformed model shape superimposed. Set the render style for both images to wireframe, and toggle off the translucency of the superimposed plot from the **Superimpose Plot Options** dialog box. Rotate the view to obtain a plot similar to that shown in *Figure 1*. By default, the deformed shape is plotted for the last increment. (For clarity, the edges of the undeformed shape are plotted using a dashed style.)

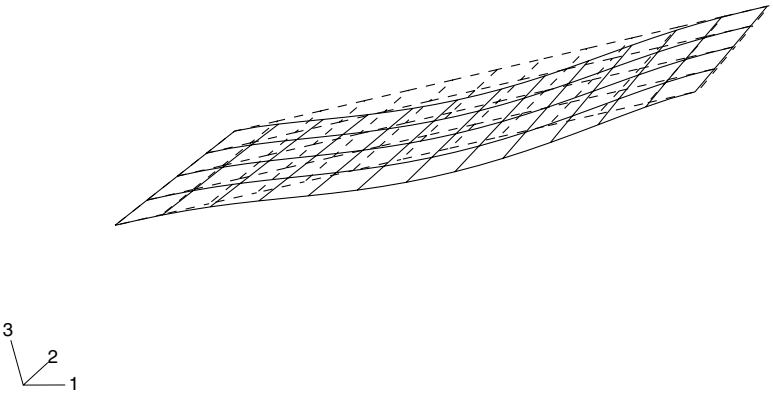


Figure 1: Deformed and undeformed model shapes of the skew plate.

Using results from other frames

You can evaluate the results from other increments saved to the output database file by selecting the appropriate frame.

1. From the main menu bar, select **Result**->**Step/Frame**.

   The **Step/Frame** dialog box appears.

2. Select `Increment 4` from the **Frame** menu.
3. Click **OK** to apply your changes and to close the **Step/Frame** dialog box.

   Any plots now requested will use results from increment 4. Repeat this procedure, substituting the increment number of interest, to move through the output database file.

   Note:

   Alternatively, you may use the **Frame Selector** dialog box to select a results frame.

X–Y plotting

You can create *X–Y* curves from either history or field data stored in the output database (`.odb`) file. *X–Y* curves can also be read from an external file or they can be typed into the Visualization module interactively.
Once curves have been created, their data can be further manipulated and plotted to the screen in graphical form.

The *X–Y* plotting capability of the Visualization module is discussed further in *Materials*.

You saved the displacements of the midspan nodes (node set `Midspan`) in the history portion of the output database file `NlSkewPlate.odb` for each increment of the simulation. You can use these results to create *X–Y* plots. In particular, you will plot the vertical displacement history of the nodes located at the edges of the plate midspan.

1. First, display only the nodes in the node set named `Midspan`: in the Results Tree, expand the **Node Sets** container underneath the output database file named `NlSkewPlate.odb`. Click mouse button 3 on the set named **MIDSPAN**, and select **Replace** from the menu that appears.
2. Use the **Common Plot Options** dialog box to show the node labels (i.e., numbers) to determine which nodes are located at the edges of the plate midspan.
3. In the Results Tree, expand the **History Output** container for the output database named `NlSkewPlate.odb`.
4. Locate the output labeled as follows: `Spatial displacement: U3 at Node xxx in NSET MIDSPAN`. Each of these curves represents the vertical motion of one of the midspan nodes.
5. Select (using **[Ctrl] + Click**) the vertical motion of the two midspan edge nodes. Use the node labels to determine which curves you need to select.
6. Click mouse button 3, and select **Plot** from the menu that appears.

   Abaqus reads the data for both curves from the output database file and plots a graph similar to the one shown in *Figure 1*. (For clarity, the second curve has been changed to a dashed line, and the default grid and legend positions have been changed.)

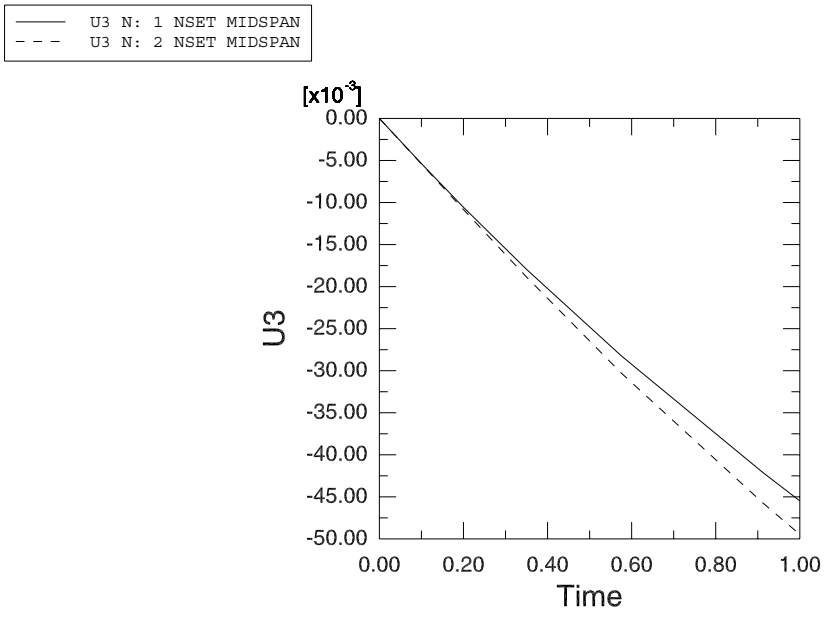


Figure 1: Midspan displacement history at the edges of the skew plate.

The nonlinear nature of this simulation is clearly seen in these curves: as the analysis progresses, the plate stiffens. In this simulation the increase in the plate stiffness with the deformation is due to membrane effects. Therefore, the resulting peak displacement is less than that predicted by the linear analysis, which did not include this effect.

Tabular data

Create a tabular data report of the midspan displacements. Use the node set `Midspan` to create the appropriate display group and the frame selector to choose the final frame. The contents of the report are shown below.

```
Field Output Report

Source 1
---------

   ODB: NlSkewPlate.odb
   Step: Apply pressure
   Frame: Increment      6: Step Time =    1.000

Loc 1 : Nodal values from source 1

Output sorted by column "Node Label".

Field Output reported at nodes for part: PLATE-1

            Node            U.U1            U.U2            U.U3
           Label         @Loc 1          @Loc 1          @Loc 1
-------------------------------------------------------------------
               1    -2.68697E-03    -747.394E-06    -49.4696E-03
               2    -2.27869E-03    -806.331E-06    -45.4817E-03
               7    -2.57405E-03    -759.298E-06    -48.5985E-03
               8    -2.49085E-03    -775.165E-06    -47.7038E-03
               9     -2.4038E-03    -793.355E-06     -46.533E-03
              66    -2.62603E-03    -750.246E-06    -49.0086E-03
              70    -2.53886E-03    -762.497E-06    -48.1876E-03
              73    -2.45757E-03    -778.207E-06     -47.144E-03
              76    -2.36229E-03    -794.069E-06    -45.9613E-03

  Minimum           -2.68697E-03    -806.331E-06    -49.4696E-03

         At Node               1               2               1
  Maximum           -2.27869E-03    -747.394E-06    -45.4817E-03

         At Node               2               1               2
```

Compare these with the displacements from the linear analysis in *Using Shell Elements*. The maximum displacement at the midspan in this simulation is about 9% less than that predicted from the linear analysis. Including the nonlinear geometric effects in the simulation reduces the vertical deflection (U3) of the midspan of the plate.

Another difference between the two analyses is that in the nonlinear simulation there are nonzero deflections in the 1- and 2-directions. What effects make the in-plane displacements, U1 and U2, nonzero in the nonlinear analysis? Why is the vertical deflection of the plate less in the nonlinear analysis?

The plate deforms into a curved shape: a geometry change that is taken into account in the nonlinear simulation. As a consequence, membrane effects cause some of the load to be carried by membrane action rather than by bending alone. This makes the plate stiffer. In addition, the pressure loading, which is always normal to the plate's surface, starts to have a component in the 1- and 2-directions as the plate deforms. The nonlinear analysis includes the effects of this stiffening and the changing direction of the pressure. Neither of these effects is included in the linear simulation.

The differences between the linear and nonlinear simulations are sufficiently large to indicate that a linear simulation is not adequate for this plate under this particular loading condition.

Running the analysis in Abaqus/Explicit

As an optional exercise, you can modify the model and run a dynamic analysis of the skew plate in Abaqus/Explicit. To do so, you must add a density of 7800 kg/m^3 to the material definition, replace the existing step with an explicit dynamic step, and change the element library to **Explicit**. After making the appropriate model changes, you can create and run a new job to examine the transient dynamic effects in the plate under a suddenly applied load.

Related Abaqus examples

Related Abaqus examples are available.

- *Elastic-plastic collapse of a thin-walled elbow under in-plane bending and internal pressure*
- *Laminated composite shells: buckling of a cylindrical panel with a circular hole*
- *Unstable static problem: reinforced plate under compressive loads*
- *Large rotation of a one degree of freedom system*
- *Vibration of a cable under tension*

Suggested reading

The following references provide additional information on nonlinear finite element methods. They allow the interested user to explore the topic in more depth.

General texts on nonlinear finite element analysis

- Belytschko, T., W. K. Liu, and B. Moran, *Nonlinear Finite Elements for Continua and Structures*, Wiley & Sons, 2000.
- Bonet, J., and R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*, Cambridge, 1997.
- Cook, R. D., D. S. Malkus, and M. E. Plesha, *Concepts and Applications of Finite Element Analysis*, Wiley & Sons, 1989.
- Crisfield, M. A., *Non-linear Finite Element Analysis of Solids and Structures, Volume I: Essentials*, Wiley & Sons, 1991.
- Crisfield, M. A., *Non-linear Finite Element Analysis of Solids and Structures, Volume II: Advanced Topics*, Wiley & Sons, 1997.
- E. Hinton (editor), *NAFEMS Introduction to Nonlinear Finite Element Analysis*, NAFEMS Ltd., 1992.
- Oden, J. T., *Finite Elements of Nonlinear Continua*, McGraw-Hill, 1972.

Summary

This section presents a summary of the concepts.

- There are three sources of nonlinearity in structural problems: material, geometric, and boundary (contact). Any combination of these may be present in an Abaqus analysis.
- Geometric nonlinearity occurs whenever the magnitude of the displacements affects the response of the structure. It includes the effects of large displacements and rotations, snap through, and load stiffening.
- In Abaqus/Standard nonlinear problems are solved iteratively using the Newton-Raphson method. A nonlinear problem will require many times the computer resources required by a linear problem.
- Abaqus/Explicit does not need to iterate to obtain a solution; however, the computational cost may be affected by reductions in the stable time increment due to large changes in geometry.
- A nonlinear analysis step is split into a number of increments.
  - Abaqus/Standard iterates to find the approximate static equilibrium obtained at the end of each new load increment. Abaqus/Standard controls the load incrementation by using convergence controls throughout the simulation.
  - Abaqus/Explicit determines a solution by advancing the kinematic state from one increment to the next, using a smaller time increment than what is commonly used in implicit analyses. The size of the increment is limited by the stable time increment. By default, time incrementation is completely automated in Abaqus/Explicit.
- The **Job Monitor** dialog box allows the progress of an analysis to be monitored while it is running. The **Job Diagnostics** dialog box contains the details of the load incrementation and iterations.
- Results can be saved at the end of each increment so that the evolution of the structure's response can be seen in the Visualization module. Results can also be plotted in the form of *X–Y* graphs.

Nonlinear Explicit Dynamics

In previous chapters you explored the basics of explicit dynamics procedures; in this chapter you will examine this topic in greater detail.
The explicit dynamics procedure can be an effective tool for solving a wide variety of nonlinear solid and structural mechanics problems. It is often complementary to an implicit solver such as Abaqus/Standard. From a user standpoint, the distinguishing characteristics of the explicit and implicit methods are:

- Explicit methods require a small time increment size that depends solely on the highest natural frequencies of the model and is independent of the type and duration of loading. Simulations generally take on the order of 10,000 to 1,000,000 increments, but the computational cost per increment is relatively small.
- Implicit methods do not place an inherent limitation on the time increment size; increment size is generally determined from accuracy and convergence considerations. Implicit simulations typically take orders of magnitude fewer increments than explicit simulations. However, since a global set of equations must be solved in each increment, the cost per increment of an implicit method is far greater than that of an explicit method.

Knowing these characteristics of the two procedures can help you decide which methodology is appropriate for your problems.

In this section:

- *Types of problems suited for Abaqus/Explicit*
- *Explicit dynamic finite element methods*
- *Automatic time incrementation and stability*
- *Example: stress wave propagation in a bar*
- *Damping of dynamic oscillations*
- *Energy balance*
- *Summary*

Types of problems suited for Abaqus/Explicit

Before discussing how the explicit dynamics procedure works, it is helpful to understand what classes of problems are well-suited to Abaqus/Explicit. Throughout this guide we have incorporated pertinent examples of the following classes of problems commonly performed in Abaqus/Explicit:

High-speed dynamic events

The explicit dynamics method was originally developed to analyze high-speed dynamic events that can be extremely costly to analyze using implicit programs, such as Abaqus/Standard. As an example of such a simulation, the effect of a short-duration blast load on a steel plate is analyzed in *Materials*. Since the load is applied rapidly and is very severe, the response of the structure changes rapidly. Accurate tracking of stress waves through the plate is important for capturing the dynamic response. Since stress waves are associated with the highest frequencies of the system, obtaining an accurate solution requires many small time increments.

Complex contact problems

Contact conditions are formulated more easily using an explicit dynamics method than using an implicit method. The result is that Abaqus/Explicit can readily analyze problems involving complex contact interaction between many independent bodies. Abaqus/Explicit is particularly well-suited for analyzing the transient dynamic response of structures that are subject to impact loads and subsequently undergo complex contact interaction within the structure. An example of such a problem is the circuit board drop test presented in *Contact*. In this example a circuit board sitting in foam packaging is dropped on the floor from a height of 1 m. The problem involves impact between the packaging and the floor, as well as rapidly changing contact conditions between the circuit board and the packaging.

Complex postbuckling problems

Unstable postbuckling problems are solved readily in Abaqus/Explicit. In such problems the stiffness of the structure changes drastically as the loads are applied. Postbuckling response often includes the effects of contact interactions.

Highly nonlinear quasi-static problems

For a variety of reasons Abaqus/Explicit is often very efficient in solving certain classes of problems that are essentially static. Quasi-static process simulation problems involving complex contact such as forging, rolling, and sheet-forming generally fall within these classes. Sheet forming problems usually include very large membrane deformations, wrinkling, and complex frictional contact conditions. Bulk forming problems are characterized by large distortions, flash formation, and contact interaction with the dies. An example of a quasi-static forming simulation is presented in *Quasi-Static Analysis with Abaqus/Explicit*.

Materials with degradation and failure

Material degradation and failure often lead to severe convergence difficulties in implicit analysis programs, but Abaqus/Explicit models such materials well. An example of material degradation is the concrete cracking model, in which tensile cracking causes the material stiffness to become negative. An example of material failure is the ductile failure model for metals, in which material stiffness can degrade until it reduces to zero. At this time the failed elements are removed from the model entirely.

Each of these types of analyses can include temperature and heat transfer effects.

Explicit dynamic finite element methods

This section contains an algorithmic description of the Abaqus/Explicit solver as well as a comparison between implicit and explicit time integration and a discussion of the advantages of the explicit dynamics method.

In this section:

- *Explicit time integration*
- *Comparison of implicit and explicit time integration procedures*
- *Advantages of the explicit time integration method*

Explicit time integration

Abaqus/Explicit uses a central difference rule to integrate the equations of motion explicitly through time, using the kinematic conditions at one increment to calculate the kinematic conditions at the next increment. At the beginning of the increment the program solves for dynamic equilibrium, which states that the nodal mass matrix, $\mathbf{M}$, times the nodal accelerations, $\ddot{\mathbf{u}}$, equals the net nodal forces (the difference between the external applied forces, $\mathbf{P}$, and internal element forces, $\mathbf{I}$):

$$\mathbf{M}\ddot{\mathbf{u}} = \mathbf{P} - \mathbf{I}.$$

The accelerations at the beginning of the current increment (time t) are calculated as

$$\ddot{\mathbf{u}}|_{(t)} = (\mathbf{M})^{-1} \cdot (\mathbf{P} - \mathbf{I})\big|_{(t)}.$$

Since the explicit procedure always uses a diagonal, or lumped, mass matrix, solving for the accelerations is trivial; there are no simultaneous equations to solve. The acceleration of any node is determined completely by its mass and the net force acting on it, making the nodal calculations very inexpensive.

The accelerations are integrated through time using the central difference rule, which calculates the change in velocity assuming that the acceleration is constant. This change in velocity is added to the velocity from the middle of the previous increment to determine the velocities at the middle of the current increment:

$$\dot{\mathbf{u}}|_{\left(t+\frac{\Delta t}{2}\right)} = \dot{\mathbf{u}}|_{\left(t-\frac{\Delta t}{2}\right)} + \frac{\left(\Delta t|_{(t+\Delta t)} + \Delta t|_{(t)}\right)}{2} \ \ddot{\mathbf{u}}|_{(t)}.$$

The velocities are integrated through time and added to the displacements at the beginning of the increment to determine the displacements at the end of the increment:

$$\mathbf{u}|_{(t+\Delta t)} = \mathbf{u}|_{(t)} + \Delta t|_{(t+\Delta t)} \dot{\mathbf{u}}\Big|_{\left(t+\frac{\Delta t}{2}\right)}.$$

Thus, satisfying dynamic equilibrium at the beginning of the increment provides the accelerations. Knowing the accelerations, the velocities and displacements are advanced “explicitly” through time. The term “explicit” refers to the fact that the state at the end of the increment is based solely on the displacements, velocities, and accelerations at the beginning of the increment. This method integrates constant accelerations exactly. For the method to produce accurate results, the time increments must be quite small so that the accelerations are nearly constant during an increment. Since the time increments must be small, analyses typically require many thousands of increments. Fortunately, each increment is inexpensive because there are no simultaneous equations to solve. Most of the computational expense lies in the element calculations to determine the internal forces of the elements acting on the nodes. The element calculations include determining element strains and applying material constitutive relationships (the element stiffness) to determine element stresses and, consequently, internal forces.

Here is a summary of the explicit dynamics algorithm:

1. Nodal calculations.

   a. Dynamic equilibrium.

   $$\ddot{\mathbf{u}}_{(t)} = \mathbf{M}^{-1}\left(\mathbf{P}_{(t)} - \mathbf{I}_{(t)}\right)$$

   b. Integrate explicitly through time.

   $$\dot{\mathbf{u}}_{\left(t+\frac{\Delta t}{2}\right)} = \dot{\mathbf{u}}_{\left(t-\frac{\Delta t}{2}\right)} + \frac{\left(\Delta t_{(t+\Delta t)} + \Delta t_{(t)}\right)}{2} \qquad \ddot{\mathbf{u}}_t$$
   $$\mathbf{u}_{(t+\Delta t)} = \mathbf{u}_{(t)} + \Delta t_{(t+\Delta t)} \dot{\mathbf{u}}_{\left(t+\frac{\Delta t}{2}\right)}.$$

2. Element calculations.

a. Compute element strain increments, $\Delta\boldsymbol{\varepsilon}$, from the strain rate, $\dot{\boldsymbol{\varepsilon}}$.

b. Compute stresses, $\boldsymbol{\sigma}$, from constitutive equations.

$$\boldsymbol{\sigma}_{(t+\Delta t)} = f\left(\boldsymbol{\sigma}_{(t)}, \Delta\boldsymbol{\varepsilon}\right)$$

c. Assemble nodal internal forces, $\mathbf{I}_{(t+\Delta t)}$.

3. Set $t + \Delta t$ to t and return to Step 1.

Comparison of implicit and explicit time integration procedures

For both the implicit and the explicit time integration procedures, equilibrium is defined in terms of the external applied forces, $\mathbf{P}$, the internal element forces, $\mathbf{I}$, and the nodal accelerations:

$$\mathbf{M}\ddot{\mathbf{u}} = \mathbf{P} - \mathbf{I},$$

where $\mathbf{M}$ is the mass matrix. Both procedures solve for nodal accelerations and use the same element calculations to determine the internal element forces. The biggest difference between the two procedures lies in the manner in which the nodal accelerations are computed. In the implicit procedure a set of linear equations is solved by a direct solution method. The computational cost of solving this set of equations is high when compared to the relatively low cost of the nodal calculations with the explicit method.

Abaqus/Standard uses automatic incrementation based on the full Newton iterative solution method. Newton's method seeks to satisfy dynamic equilibrium at the end of the increment at time $t + \Delta t$ and to compute displacements at the same time. The time increment, Δt, is relatively large compared to that used in the explicit method because the implicit scheme is unconditionally stable. For a nonlinear problem each increment typically requires several iterations to obtain a solution within the prescribed tolerances. Each Newton iteration solves for a correction, $\mathbf{c}_j$, to the incremental displacements, $\Delta\mathbf{u}_j$. Each iteration requires the solution of a set of simultaneous equations,

$$\widehat{\mathbf{K}}_j\mathbf{c}_j = \mathbf{P}_j - \mathbf{I}_j - \mathbf{M}_j\ddot{\mathbf{u}}_j,$$

which is an expensive procedure for large models. The effective stiffness matrix, $\widehat{\mathbf{K}}_j$, is a linear combination of the tangent stiffness matrix and the mass matrix for the iteration. The iterations continue until several quantities—force residual, displacement correction, etc.—are within the prescribed tolerances. For a smooth nonlinear response Newton's method has a quadratic rate of convergence, as illustrated below:

Iteration	Relative Error
1	1
2	10^{-2}
3	10^{-4}
.	.
.	.
.	.

However, if the model contains highly discontinuous processes, such as contact and frictional sliding, quadratic convergence may be lost and a large number of iterations may be required. Cutbacks in the time increment size may become necessary to satisfy equilibrium. In extreme cases the resulting time increment size in the implicit analysis may be on the same order as a typical stable time increment for an explicit analysis, while still carrying the high solution cost of implicit iteration. In some cases convergence may not be possible using the implicit method.

Each iteration in an implicit analysis requires solving a large system of linear equations, a procedure that requires considerable computation, disk space, and memory. For large problems these equation solver requirements are dominant over the requirements of the element and material calculations, which are similar for an analysis in Abaqus/Explicit. As the problem size increases, the equation solver requirements grow rapidly so that, in practice, the maximum size of an implicit analysis that can be solved on a given machine often is dictated by the amount of disk space and memory available on the machine rather than by the required computation time.

Advantages of the explicit time integration method

The explicit method is especially well-suited to solving high-speed dynamic events that require many small increments to obtain a high-resolution solution. If the duration of the event is short, the solution can be obtained efficiently.

Contact conditions and other extremely discontinuous events are readily formulated in the explicit method and can be enforced on a node-by-node basis without iteration. The nodal accelerations can be adjusted to balance the external and internal forces during contact.

The most striking feature of the explicit method is the absence of a global tangent stiffness matrix, which is required with implicit methods. Since the state of the model is advanced explicitly, iterations and tolerances are not required.

Automatic time incrementation and stability

The stability limit dictates the maximum time increment used by the Abaqus/Explicit solver. It is a critical factor in the performance of Abaqus/Explicit. The following sections describe the stability limit and discuss how Abaqus/Explicit determines this value.
Issues surrounding the model design parameters that affect the stability limit are also addressed. These model parameters include the model mass, material, and mesh.

In this section:

- *Conditional stability of the explicit method*
- *Definition of the stability limit*
- *Fully automatic time incrementation versus fixed time incrementation in Abaqus/Explicit*
- *Mass scaling to control time incrementation*
- *Effect of material on stability limit*
- *Effect of mesh on stability limit*
- *Numerical instability*

Conditional stability of the explicit method

With the explicit method the state of the model is advanced through an increment of time, Δt, based on the state of the model at the start of the increment at time t. The amount of time that the state can be advanced and still remain an accurate representation of the problem is typically quite short. If the time increment is larger than this maximum amount of time, the increment is said to have exceeded the *stability limit*. A possible effect of exceeding the stability limit is a numerical instability, which may lead to an unbounded solution. It generally is not possible to determine the stability limit exactly, so conservative estimates are used instead. The stability limit has a great effect on reliability and accuracy, so it must be determined consistently and conservatively. For computational efficiency Abaqus/Explicit chooses the time increments to be as close as possible to the stability limit without exceeding it.

Definition of the stability limit

The stability limit is defined in terms of the highest frequency in the system ($\omega_{\max}$). Without damping the stability limit is defined by the expression

$$\Delta t_{\text{stable}} = \frac{2}{\omega_{\max}},$$

and with damping it is defined by the expression

$$\Delta t_{\text{stable}} = \frac{2}{\omega_{\max}} \left(\sqrt{1+\xi^2} - \xi \right),$$

where ξ is the fraction of critical damping in the mode with the highest frequency. (Recall that critical damping defines the limit between oscillatory and non-oscillatory motion in the context of free-damped vibration. Abaqus/Explicit always introduces a small amount of damping in the form of bulk viscosity to control high-frequency oscillations.) Perhaps contrary to engineering intuition, damping always reduces the stability limit.

The actual highest frequency in the system is based on a complex set of interacting factors, and it is not computationally feasible to calculate its exact value. Alternately, we use a simple estimate that is efficient and conservative. Instead of looking at the global model, we estimate the highest frequency of each individual element in the model, which is always associated with the dilatational mode. It can be shown that the highest element frequency determined on an element-by-element basis is always higher than the highest frequency in the assembled finite element model.

Based on the element-by-element estimate, the stability limit can be redefined using the element length, L^e, and the wave speed of the material, c_d:

$$\Delta t_{\text{stable}} = \frac{L^e}{c_d}.$$

For most element types—a distorted quadrilateral element, for example—the above equation is only an estimate of the actual element-by-element stability limit because it is not clear how the element length should be determined. As an approximation the shortest element distance can be used, but the resulting estimate is not always conservative. The shorter the element length, the smaller the stability limit. The wave speed is a property of the material. For a linear elastic material with a Poisson's ratio of zero

$$c_d = \sqrt{\frac{E}{\rho}},$$

where E is Young's modulus and ρ is the mass density. The stiffer the material, the higher the wave speed, resulting in a smaller stability limit. The higher the density, the lower the wave speed, resulting in a larger stability limit.

Our simplified stability limit definition provides some intuitive understanding. The stability limit is the transit time of a dilatational wave across the distance defined by the characteristic element length. If we know the size of the smallest element dimension and the wave speed of the material, we can estimate the stability limit. For example, if the smallest element dimension is 5 mm and the dilatational wave speed is 5000 m/s, the stable time increment is on the order of 1×10^{-6} s.

Fully automatic time incrementation versus fixed time incrementation in Abaqus/Explicit

Abaqus/Explicit uses equations such as those discussed in the previous section to adjust the time increment size throughout the analysis so that the stability limit, based on the current state of the model, is never exceeded. Time incrementation is automatic and requires no user intervention, not even a suggested initial time increment. The stability limit is a mathematical concept resulting from the numerical model. Since the finite element program has all of the relevant details, it can determine an efficient and conservative stability limit. However, Abaqus/Explicit does allow the user to override the automatic time incrementation, if desired.

The time increment used in an explicit 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.

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 may switch to the global estimation method under certain circumstances.

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 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.

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 time increment specified directly by the user. Fixed time incrementation may 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 may be used. When fixed time incrementation is used, Abaqus/Explicit will not check that the computed response is stable during the step. The user should ensure that a valid response has been obtained by carefully checking the energy history and other response variables.

Mass scaling to control time incrementation

Since the mass density influences the stability limit, under some circumstances scaling the mass density can potentially increase the efficiency of an analysis. For example, because of the complex discretization of many models, there are often regions containing very small or poorly shaped elements that control the stability limit. These controlling elements are often few in number and may exist in localized areas. By increasing the mass of only these controlling elements, the stability limit can be increased significantly, while the effect on the overall dynamic behavior of the model may be negligible.

The automatic mass scaling features in Abaqus/Explicit can keep offending elements from hindering the stability limit. There are two fundamental approaches used in mass scaling: defining a scaling factor directly or defining a desired element-by-element stable time increment for the elements whose mass is to be scaled. These two approaches, described in detail in *Mass Scaling*, permit additional user control over the stability limit. However, use caution when employing mass scaling since significantly changing the mass of the model may change the physics of the problem.

Effect of material on stability limit

The material model affects the stability limit through its effect on the dilatational wave speed. In a linear material the wave speed is constant; therefore, the only changes in the stability limit during the analysis result from changes in the smallest element dimension during the analysis. In a nonlinear material, such as a metal with plasticity, the wave speed changes as the material yields and the stiffness of the material changes. Abaqus/Explicit monitors the effective wave speeds in the model throughout the analysis, and the current material state in each element is used for stability estimates. After yielding, the stiffness decreases, reducing the wave speed and, consequently, increasing the stability limit.

Effect of mesh on stability limit

Since the stability limit is roughly proportional to the shortest element dimension, it is advantageous to keep the element size as large as possible. Unfortunately, for accurate analyses a fine mesh is often necessary. To obtain the highest possible stability limit while using the required level of mesh refinement, the best approach is to have a mesh that is as uniform as possible. Since the stability limit is based on the smallest element dimension in the model, even a single small or poorly shaped element can reduce the stability limit drastically. For diagnostic purposes Abaqus/Explicit provides a list in the status (`.sta`) file of the 10 elements in the mesh with the lowest stability limit. If the model contains some elements whose stability limits are much lower than those of the rest of the mesh, remeshing the model more uniformly may be worthwhile.

Numerical instability

Abaqus/Explicit remains stable for most elements under most circumstances. It is possible, however, to define spring and dashpot elements such that they become unstable during the course of an analysis. Therefore, it is useful to be able to recognize a numerical instability if it occurs in your analysis. If it does occur, the result typically will be unbounded, nonphysical, and often characterized by oscillatory solutions.

Example: stress wave propagation in a bar

This example demonstrates some of the fundamental ideas in explicit dynamics and illustrates stability limits and the effect of mesh refinement and material properties on the solution time.

In this section:

- *Stress wave propagation analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*
- *How the mesh affects the stable time increment and CPU time*
- *How the material affects the stable time increment and CPU time*

Stress wave propagation analysis

In this example you will create a model of a stress wave propagation in a bar.

The bar has the dimensions shown in *Figure 1*.

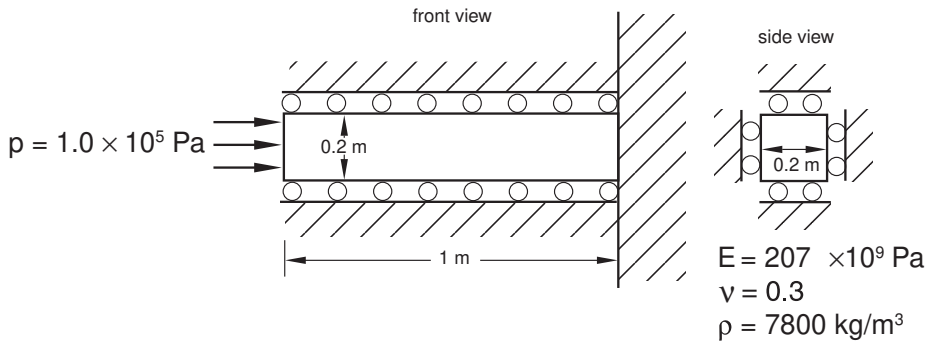


Figure 1: Schematic for wave propagation in a bar.

To make the problem a one-dimensional strain problem, all four lateral faces are on rollers; thus, the three-dimensional model simulates a one-dimensional problem. The material is steel with the properties shown in *Figure 1*. The free end of the bar is subjected to a blast load with a magnitude of 1.0×10^5 Pa and a duration of 3.88×10^{-5} s. The normalized load versus time is shown in *Figure 2*.

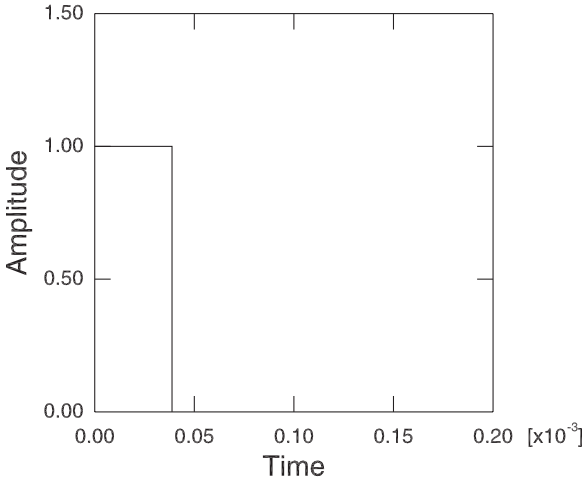


Figure 2: Blast amplitude versus time.

Additional information

• *Abaqus Basics*

Preprocessing—creating the model with Abaqus/CAE

In this section we discuss how to use Abaqus/CAE to create the model for this simulation. Abaqus provides scripts that replicate the complete analysis model for this problem.
Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Stress wave propagation in a bar*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Stress wave propagation in a bar**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

In this section:

- *Defining the model geometry*
- *Defining the material and section properties*
- *Creating an assembly*
- *Creating geometry sets and surfaces*
- *Defining steps*
- *Specifying output requests*
- *Prescribing boundary conditions*
- *Defining the load history*
- *Creating a mesh*
- *Creating, running, and monitoring a job*
- *The status (.sta) file*

Defining the model geometry

In this example you will create a three-dimensional, deformable body with a solid, extruded base feature. You will first sketch the two-dimensional profile of the bar and then extrude it.

1. Create a new part named `Bar`, and accept the default settings of a three-dimensional, deformable body and a solid, extruded base feature in the **Create Part** dialog box. Use an approximate size of `0.5` for the model.
2. Use the dimensions given in *Figure 1* to sketch the cross-section of the bar.

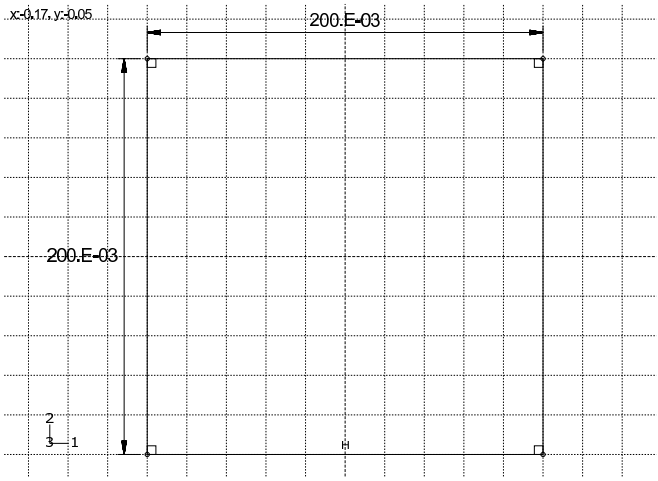


Figure 1: Rectangular cross-section (with grid spacing doubled).

The following possible approach can be used:

a. Create a rectangle centered about the origin using the **Create Lines: Rectangle** tool.



Tip: Create fixed horizontal and vertical construction lines through the origin of the sketch plane, and apply appropriate geometry constraints to the sketch.

b. Dimension the sketch so that the cross-section is 0.20 m high × 0.20 m wide.

c. Click **Done** in the prompt area when you are finished sketching the profile.

The **Edit Base Extrusion** dialog box appears. To complete the part definition, you must specify the distance over which the profile will be extruded.

d. In the dialog box, enter an extrusion depth of `1.0` m.

3. Save your model in a model database file named `Bar.cae`.

Defining the material and section properties

You can define the material and section properties.

Create a single linear elastic material named `Steel` with a mass density of `7800` kg/m^3, a Young's modulus of `207E9` Pa, and a Poisson's ratio of `0.3`.

Create a solid, homogeneous section definition named `BarSection` with **Steel** as the material.

Assign the section definition `BarSection` to the entire part.

Creating an assembly

You can create the assembly.

Create an instance of the bar. The model is oriented by default so that the global 3-axis lies along the length of the bar.

Creating geometry sets and surfaces

Next, you can create the geometry sets and surfaces.

Create the geometry sets `TOP`, `BOT`, `FRONT`, `BACK`, `FIX`, and `OUT` as shown in *Figure 1*. (The set `OUT` contains the edge shown in bold in *Figure 1*.) Create the surface named `LOAD` shown in *Figure 2*. These regions will be used later for the application of loads and boundary conditions, as well as for the definition of output requests.

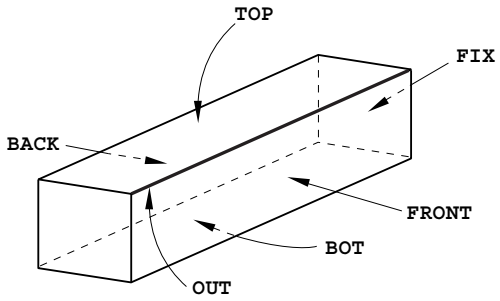


Figure 1: Sets.

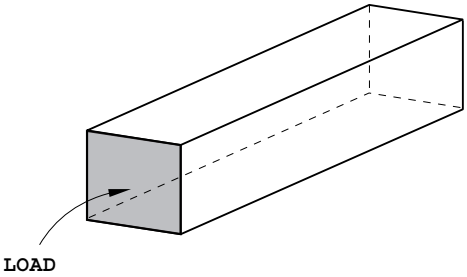


Figure 2: Surface.

Defining steps

You define an explicit step for this analysis.

Create a single dynamic, explicit step named `BlastLoad`. Enter `Apply pressure load pulse` as the step description, and set the **Time period** to `2.0E-4` s. In the **Edit Step** dialog box, click the **Other** tab. To keep the stress wave as sharp as possible, set the **Quadratic bulk viscosity parameter** (discussed in *Bulk viscosity*) to zero.

Specifying output requests

You can specify field and history output requests.

Edit the default field output request so that preselected field data are written to the output database at four equally spaced intervals for the step `BlastLoad`.

Delete the existing default history output request, and create a new set of history output requests. In the **Create History Output** dialog box, accept the default name of `H-Output-1` and select the `BlastLoad` step. Click **Continue**. Click the arrow next to the **Domain** field, select **Set**, and select `OUT`. In the **Output Variables** section, click on the triangle to the left of **Stresses**. Click on the triangle to the left of **S, Stress components and invariants**, and toggle on the **S33** variable, which is the component of stress in the axial direction of the bar. Specify that output be saved at every increment.

Prescribing boundary conditions

You prescribe several boundary conditions.

Create a boundary condition named `Fix right end`, and constrain the right end of the bar (geometry set `FIX`) in all three directions (see *Figure 1*). Create additional boundary conditions to constrain the top, bottom, front, and back faces in the directions normal to these faces (the 1-direction for sets `FRONT` and `BACK` and the 2-direction for sets `TOP` and `BOT`).

Defining the load history

The blast load is applied at its maximum value instantaneously and is held constant for 3.88×10^{-5} s. Then the load is suddenly removed and held constant at zero.

Create an amplitude definition named `Blast` using the data shown in *Figure 1*. For this problem the pressure load at any given time is the specified magnitude of the pressure load times the value interpolated from the amplitude curve.

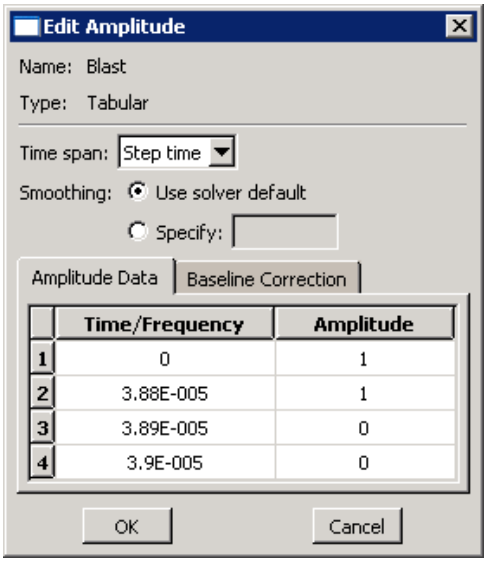


Figure 1: Tabular data for the blast load amplitude definition.

Create a pressure load named `Blast load`, and select `BlastLoad` as the step in which it will be applied. Apply the load to the surface `LOAD`. Select **Uniform** for the distribution, specify a value of `1.0E5` Pa for the load magnitude, and select **Blast** for the amplitude.

Creating a mesh

Next, you can create a mesh.

Using the material properties (neglecting Poisson's ratio), we can calculate the wave speed of the material using the equations introduced earlier.

$$c_d = \sqrt{\frac{E}{\rho}} = \sqrt{\frac{207 \times 10^9 \ \text{Pa}}{7800 \ \text{kg/m}^3}} = 5.15 \times 10^3 \ \text{m/s}.$$

At this speed, the wave passes to the fixed end of the bar in 1.94×10^{-4} s. Since we are interested in stress propagation along the length of the bar through time, we need a sufficiently refined mesh to capture the stress wave accurately. We will assume that the blast load will take place over the span of 10 elements. To determine the length of these 10 elements, multiply the blast duration by the wave speed:

$$L_{10el} = \left(3.88 \times 10^{-5}\text{s}\right) c_d.$$

The length of 10 elements is 0.2 m. Since the total length of the bar is 1.0 m, we would have 50 elements along the length. To keep the mesh uniform, we will also have 10 elements in each of the transverse directions, making the mesh $50 \times 10 \times 10$. This mesh is shown in *Figure 1*.

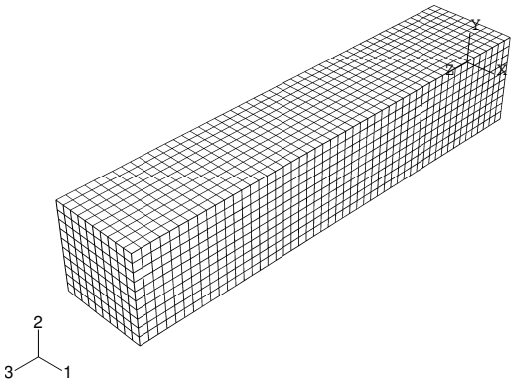


Figure 1: 50 × 10 × 10 mesh.

Seed the part with a target global element size of `0.02`. Select C3D8R from the **Explicit** library as the element type, and mesh the part.

Note:

The proposed mesh density exceeds the model size limits of the Abaqus Learning Edition. If you are using this product, specify one of the following:

- A global element size of 0.04.
- A $50 \times 3 \times 3$ mesh.
- A $25 \times 5 \times 5$ mesh.

Creating, running, and monitoring a job

Create a job named `Bar`, and enter `Stress wave propagation in a bar (SI units)` for the job description. Submit the job, and monitor the results. If any errors are encountered, correct the model and rerun the simulation. Be sure to investigate the cause of any warning messages and take appropriate action; recall that some warning messages can be safely ignored while others require corrective action.

The status (.sta) file

You can also look at the status file, `Bar.sta`, to monitor the job progress.

You can browse or search this file by clicking the **Status File** tab in the **Job Monitor**. The status file contains information about moments of inertia, followed by information concerning the stability limit. The 10 elements with the lowest stable time limits are listed in rank order.

```
    Most critical elements:
    Element number   Rank    Time increment   Increment ratio
    (Instance name)
   -------------------------------------------------------------
           1          1       2.237266E-06      1.000000E+00
    BAR-1
          10          2       2.237266E-06      1.000000E+00
    BAR-1
          21          3       2.237266E-06      1.000000E+00
    BAR-1
          30          4       2.237266E-06      1.000000E+00
    BAR-1
          31          5       2.237266E-06      1.000000E+00
    BAR-1
          40          6       2.237266E-06      1.000000E+00
    BAR-1
          51          7       2.237266E-06      1.000000E+00
    BAR-1
          60          8       2.237266E-06      1.000000E+00
    BAR-1
          61          9       2.237266E-06      1.000000E+00
    BAR-1
          70         10       2.237266E-06      1.000000E+00
    BAR-1
```

The status file continues with information about the progress of the solution.

```
   STEP 1  ORIGIN 0.0000

   Total memory used for step 1 is approximately 6.8 megabytes.
   Global time estimation algorithm will be used.
   Scaling factor :  1.0000
   Variable mass scaling factor at zero increment:  1.0000


              STEP     TOTAL       CPU      STABLE    CRITICAL    KINETIC
 TOTAL
INCREMENT     TIME      TIME      TIME   INCREMENT     ELEMENT     ENERGY
ENERGY
        0  0.000E+00 0.000E+00  00:00:00 2.237E-06          70  0.000E+00
0.000E+00
INSTANCE WITH CRITICAL ELEMENT: BAR-1
ODB Field Frame Number      0 of      4 requested intervals at increment zero.
        5  1.119E-05 1.119E-05  00:00:00 2.237E-06          70  4.504E-05
-1.963E-06
       10  2.237E-05 2.237E-05  00:00:00 2.237E-06        2010  9.189E-05
-2.218E-06
       15  3.471E-05 3.471E-05  00:00:00 2.931E-06        2090  1.434E-04
-2.361E-06
       19  4.640E-05 4.640E-05  00:00:00 2.910E-06        1040  1.607E-04
1.850E-06
       21  5.221E-05 5.221E-05  00:00:00 2.902E-06         600  1.588E-04
2.585E-06
```

```
ODB Field Frame Number      1 of      4 requested intervals at  5.176230E-05
       25  6.380E-05 6.380E-05  00:00:00 2.890E-06         590  1.569E-04
6.082E-07
       29  7.535E-05 7.535E-05  00:00:00 2.880E-06        4640  1.553E-04
-7.049E-07
       33  8.685E-05 8.685E-05  00:00:00 2.873E-06        4670  1.537E-04
-1.386E-06
```

Similar information is available in the **Job Diagnostics** dialog box of the Visualization module. This dialog box also allows you to plot histories of the data, as shown in *Figure 1*. To create a history plot, select a step from the **Job History**, select a column in the **Incrementation** tab, and click **Plot selected column**.

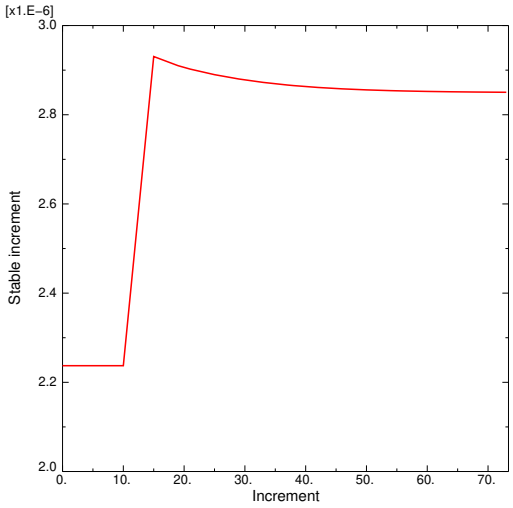


Figure 1: Stable time increment history.

Postprocessing

In the Model Tree, click mouse button 3 on the job named **Bar** and select **Results** from the menu that appears to enter the Visualization module and automatically open the output database (`.odb`) file created by this job. Alternatively, from the **Module** list located in the context bar, select **Visualization** to enter the Visualization module; open the `.odb` file by selecting **File->Open** from the main menu bar and double-clicking on the appropriate file.

Plotting the stress along a path

We are interested in looking at how the stress distribution along the length of the bar changes with time. To do so, we will look at the stress distribution at three different times throughout the course of the analysis.

Create a curve of the variation of the stress in the 3-direction (S33) along the axis of the bar for each of the first three frames of the output database file. To create these plots, you first need to define a straight path along the axis of the bar.

To create a point list path along the center of the bar:

1. In the Results Tree, double-click **Paths**.

   The **Create Path** dialog box appears.

2. Name the path `Center`. Select **Point list** as the path type, and click **Continue**.

   The **Edit Point List Path** dialog box appears.

3. In the **Point Coordinates** table, enter the coordinates of the centers of both ends of the bar. The input specifies a path from the first point to the second point, as defined in the global coordinate system of the model.

   Note:

   If you generated the geometry and mesh using the procedure described earlier, the table entries are `0, 0, 1` and `0, 0, 0`. If you used an alternate procedure to generate the bar geometry, you can use the  tool in the Query toolset to determine the coordinates of the centers at each end of the bar.

4. When you have finished, click **OK** to close the **Edit Point List Path** dialog box.

To save X–Y plots of stress along the path at three different times:

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.

2. Choose **Path** as the *X–Y* data source, and click **Continue**.

   The **XY Data from Path** dialog box appears with the path that you created visible in the list of available paths. If the undeformed model shape is currently displayed, the path you select is highlighted in the plot.

3. Toggle on **Include intersections** under **Point Locations**.
4. Accept **True distance** as the selection in the **X Values** region of the dialog box.
5. Click **Field Output** in the **Y Values** region of the dialog box to open the **Field Output** dialog box.
6. Select the S33 stress component, and click **OK**.

   The field output variable in the **XY Data from Path** dialog box changes to indicate that stress data in the 3-direction (S33) will be created.



Note:

Abaqus/CAE may warn you that the field output variable will not affect the current image. Leave the plot state **As is**, and click **OK** to continue.

7. Click **Step/Frame** in the **Y Values** region of the **XY Data from Path** dialog box.
8. In the **Step/Frame** dialog box that appears, choose frame 1, which is the second of the five recorded frames. (The first frame listed, frame 0, is the base state of the model at the beginning of the step.) Click **OK**.

   The **Y Values** region of the **XY Data from Path** dialog box changes to indicate that data from Step 1, frame 1 will be created.
9. To save the *X*–*Y* data, click **Save As**.

   The **Save XY Data As** dialog box appears.
10. Name the *X*–*Y* data `S33_T1`, and click **OK**.

    `S33_T1` appears in the **XYData** container of the Results Tree.
11. Repeat Steps 7 through 9 to create *X*–*Y* data for frames 2 and 3. Name the data sets `S33_T2` and `S33_T3`, respectively.
12. To close the **XY Data from Path** dialog box, click **Cancel**.

To plot the stress curves:

1. In the **XYData** container, drag the cursor to select all three *X*–*Y* data sets.
2. Click mouse button 3, and select **Plot** from the menu that appears.

   Abaqus/CAE plots the stress in the 3-direction along the center of the bar for frames 1, 2, and 3, corresponding to approximate simulation times of 5×10^{-5} s, 1×10^{-4} s, and 1.5×10^{-4} s, respectively.
3. Click  in the prompt area to cancel the current procedure.

To customize the X–Y plot:

1. Double-click the *Y*-axis.

   The **Axis Options** dialog box appears. The **Y Axis** is selected.
2. In the **Tick Mode** region of the **Scale** tabbed page, select **By increment** and specify that the *Y*-axis major tick marks occur at `20E3` Pa increments.

   You can also customize the axis titles.
3. Switch to the **Title** tabbed page.
4. Enter `Stress - S33 (Pa)` as the *Y*-axis title.
5. To edit the *X*-axis, select the axis label in the **X Axis** field of the dialog box. In the **Title** tabbed page of the dialog box, enter `Distance along bar (m)` as the *X*-axis title.
6. Click **Dismiss** to close the **Axis Options** dialog box.

To customize the appearance of the curves in the X–Y plot:

1. In the Visualization toolbox, click  to open the **Curve Options** dialog box.

2. In the **Curves** field, select `S33_T2`.
3. Choose the dotted line style for the `S33_T2` curve.

   The `S33_T2` curve becomes dotted.
4. Repeat Steps 2 and 3 to make the `S33_T3` curve dashed.
5. Select each of the curves and, in the **Attributes** field, select **Name** as the default legend text.
6. Dismiss the **Curve Options** dialog box.

   The customized plot appears in *Figure 1*. (For clarity, the default grid and legend positions have been changed.)

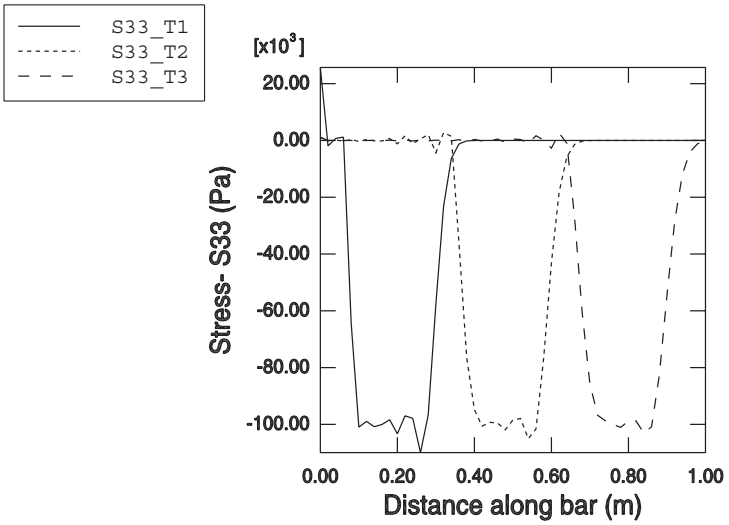


Figure 1: Stress (S33) along the bar at three different time instances.

We can see that the length of the bar affected by the stress wave is approximately 0.2 m in each of the three curves. This distance should correspond to the distance that the blast wave travels during its time of application, which can be checked by a simple calculation. If the length of the wave front is 0.2 m and the wave speed is 5.15×10^3 m/s, the time it takes for the wave to travel 0.2 m is 3.88×10^{-5} s. As expected, this is the duration of the blast load that we applied. The stress wave is not exactly square as it passes along the bar. In particular, there is "ringing" or oscillation of the stress behind the sudden changes in stress. Linear bulk viscosity, discussed later in this chapter, damps the ringing so that it does not affect the results adversely.

Creating a history plot

Another way to study the results is to view the time history of stress at three different points within the bar; for example, at distances of 0.25 m, 0.50 m, and 0.75 m from the loaded end of the bar. To do this, we must first determine the labels of the elements located at these positions. An easy way to accomplish this is to query the elements in a display group consisting of the elements along the edge of the bar (set `OUT`).

To query the element labels:

1. In the Results Tree, expand the **Element Sets** container underneath the output database file named `Bar.odb`. Click mouse button 3 on the set named **OUT**, and select **Replace** from the menu that appears.
2. From the main menu bar, select **Tools->Query**; or use the  tool in the Query toolset.
3. In the **Query** dialog box that appears, click **Element**.

4. Click on the elements shaded in *Figure 2* (every 13th element along the bar). The element ID (label) appears in the prompt area (and also in the message area). Make note of the element labels for the three shaded elements.

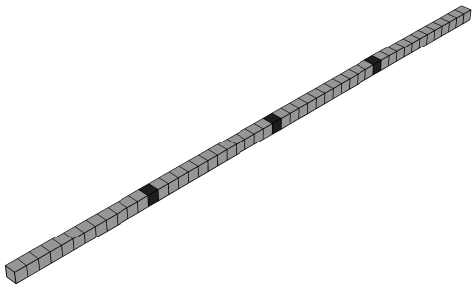


Figure 2: `History plot` display group.

To plot the stress history:

1. In the Results Tree, click mouse button 3 on **History Output** and deselect **Group Children** from the menu that appears.
2. Select the data for the three elements you have identified (again, every 13th element). Use **[Ctrl] + Click** to select multiple *X*–*Y* data sets.
3. Click mouse button 3, and select **Plot** from the menu that appears.

   Abaqus/CAE displays an *X*–*Y* plot of the longitudinal stress in each element versus time.
4. Click  in the prompt area to cancel the current procedure.

As before, you can customize the appearance of the plot.

To customize the X–Y plot:

1. Double-click the *X*-axis.

   The **Axis Options** dialog box appears.
2. Switch to the **Title** tabbed page.
3. Specify `Total time (s)` as the *X*-axis title.
4. Click **Dismiss** to close the dialog box.

To customize the appearance of the curves in the X–Y plot:

1. In the Visualization toolbox, click  to open the **Curve Options** dialog box.
2. In the **Curves** field, select the temporary *X*–*Y* data label that corresponds to the element closest to the free end of the bar. (Of the elements in this set, this one is affected first by the stress wave.)
3. Enter `S33-0.25` as the curve legend text.
4. In the **Curves** field, select the temporary *X*–*Y* data label that corresponds to the element in the middle of the bar. (This is the element affected next by the stress wave.)

5. Specify `S33-0.5` as the curve legend text, and change the curve style to dotted.
6. In the **Curves** field, select the temporary *X–Y* data label that corresponds to the element closest to the fixed end of the bar. (This is the element affected last by the stress wave.)
7. Specify `S33-0.75` as the curve legend text, and change the curve style to dashed.
8. Click **Dismiss** to close the dialog box.

   The customized plot appears in *Figure 3*. (For clarity, the default grid and legend positions have been changed.)

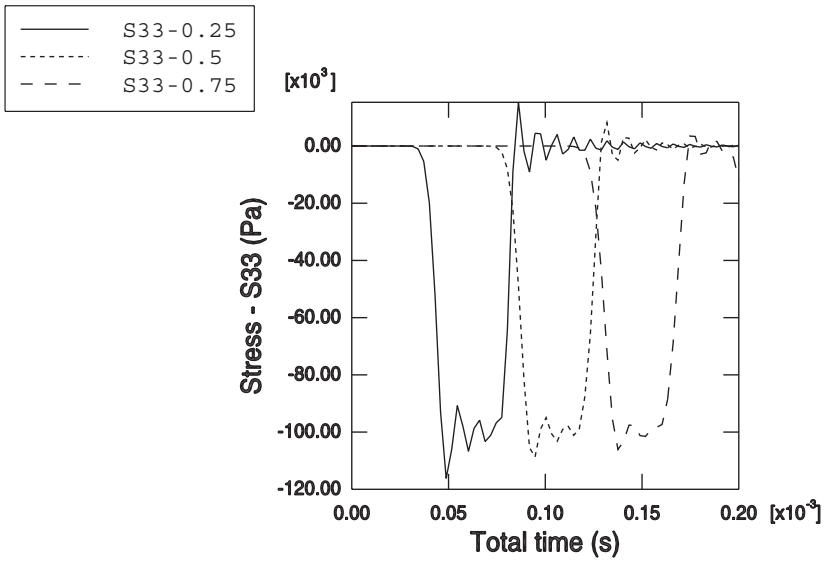


Figure 3: Time history of stress (S33) at three points along the length of the bar (0.25 m, 0.5 m, and 0.75 m).

In the history plot we can see that stress at a given point increases as the stress wave travels through the point. Once the stress wave has passed completely through the point, the stress at the point oscillates about zero.

How the mesh affects the stable time increment and CPU time

In *Automatic time incrementation and stability* we discussed how mesh refinement affects the stability limit and the CPU time. Here we will illustrate this effect with the wave propagation problem. We began with a reasonably refined mesh of square elements with 50 elements along the length and 10 elements in each of the two transverse directions. For illustrative purposes, we will now use a coarse mesh of 25 × 5 × 5 elements and observe how refining the mesh in the various directions changes the CPU time. The four meshes are shown in *Figure 1*.

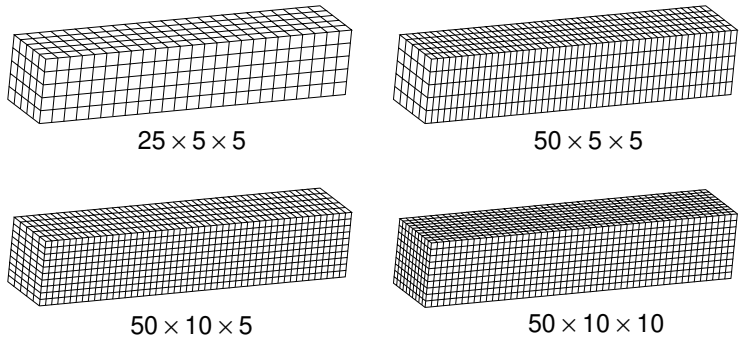


Figure 1: Meshes from least to most refined.

Table 1 shows how the CPU time (normalized with respect to the coarse mesh model result) changes with mesh refinement for this problem. The first half of the table provides the expected results, based on the simplified stability equations presented in this guide; the second half of the table provides the results obtained by running the analyses in Abaqus/Explicit on a desktop workstation.

Table 1: Mesh refinement and solution time.

Mesh	Simplified Theory			Actual		
	Δt_{stable} (s)	Number of Elements	CPU Time (s)	Max Δt_{stable} (s)	Number of Elements	Normalized CPU Time
25 × 5 × 5	A	B	C	5.754E-06	625	1
50 × 5 × 5	A/2	2B	4C	2.954E-06	1250	4
50 × 10 × 5	A/2	4B	8C	2.933E-06	2500	8.33
50 × 10 × 10	A/2	8B	16C	2.907E-06	5000	16.67

For the theoretical results we choose the coarsest mesh, 25 × 5 × 5, as the base state, and we define the stable time increment, the number of elements, and the CPU time as variables A, B, and C, respectively. As the mesh is refined, two things happen: the smallest element dimension decreases, and the number of elements in the mesh increases. Each of these effects increases the CPU time. In the first level of refinement, the 50 × 5 × 5 mesh, the smallest element dimension is cut in half and the number of elements is doubled, increasing the CPU time by a factor of four over the previous mesh. However, further doubling the mesh to 50 × 10 × 5 does not change the smallest element dimension; it only doubles the number of elements. Therefore, the CPU time increases by only a factor of two over the 50 × 5 × 5 mesh. Further refining the mesh so that the elements are uniform and square in the 50 × 10 × 10 mesh again doubles the number of elements and the CPU time.

This simplified calculation predicts quite well the trends of how mesh refinement affects the stable time increment and CPU time. However, there are reasons why we did not compare the predicted and actual stable time increment values. First, recall that we made the approximation that the stable time increment is

$$\Delta t_{\text{stable}} = \frac{L^e}{c_d}.$$

We then assumed that the characteristic element length, L^e, is the smallest element dimension, whereas Abaqus/Explicit actually determines the characteristic element length based on the overall size and shape of the element. Another complication is that Abaqus/Explicit employs a global stability estimator, which allows a larger stable time increment to be used. These factors make it difficult to predict the stable time increment accurately before running the analysis. However, since the trends follow nicely from the simplified theory, it is straightforward to predict how the stable time increment will change with mesh refinement.

How the material affects the stable time increment and CPU time

The same wave propagation analysis performed on different materials would take different amounts of CPU time, depending on the wave speed of the material. For example, if we were to change the material from steel to aluminum, the wave speed would change from 5.15×10^3 m/s to

$$c_d = \sqrt{\frac{E}{\rho}} = \sqrt{\frac{70 \times 10^9 \quad \text{Pa}}{2700 \quad \text{kg/m}^3}} = 5.09 \times 10^3 \quad \text{m/s.}$$

The change from aluminum to steel has minimal effect on the stable time increment, because the stiffness and the density differ by nearly the same amount. In the case of lead the difference is more substantial, as the wave speed decreases to

$$c_d = \sqrt{\frac{E}{\rho}} = \sqrt{\frac{14 \times 10^9 \quad \text{Pa}}{11240 \quad \text{kg/m}^3}} = 1.12 \times 10^3 \quad \text{m/s,}$$

which is approximately one-fifth the wave speed of steel. The stable time increment for the lead bar would be five times the stable time increment of our steel bar.

Damping of dynamic oscillations

There are two reasons for adding damping to a model: to limit numerical oscillations or to add physical damping to the system. Abaqus/Explicit provides several methods of introducing damping into the analysis.

In this section:

- *Bulk viscosity*
- *Viscous pressure*
- *Material damping*
- *Discrete dashpots*

Bulk viscosity

Bulk viscosity introduces damping associated with volumetric straining. Its purpose is to improve the modeling of high-speed dynamic events. Abaqus/Explicit contains linear and quadratic forms of bulk viscosity. You can modify the default bulk viscosity parameters in the step definition, although it is rarely necessary to do so. The bulk viscosity pressure is not included in the material point stresses because it is intended as a numerical effect only. As such, it is not considered part of the material's constitutive response.

Linear bulk viscosity

By default, linear bulk viscosity is always included to damp “ringing” in the highest element frequency. It generates a bulk viscosity pressure that is linear in the volumetric strain rate, according to the following equation:

$$p_1 = b_1 \rho c_d L^e \dot{\varepsilon}_{vol},$$

where b_1 is a damping coefficient, whose default value is 0.06, ρ is the current material density, c_d is the current dilatational wave speed, L^e is the element characteristic length, and $\dot{\varepsilon}_{vol}$ is the volumetric strain rate.

Quadratic bulk viscosity

Quadratic bulk viscosity is included only in continuum elements (except for the plane stress element, CPS4R) and is applied only if the volumetric strain rate is compressive. The bulk viscosity pressure is quadratic in the strain rate, according to the following equation:

$$p_2 = \rho (b_2 L^e)^2 \, |\dot{\varepsilon}_{vol}| \, min\,(0, \dot{\varepsilon}_{vol}),$$

where b_2 is the damping coefficient, whose default value is 1.2.

The quadratic bulk viscosity smears 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, as shown in *Figure 1*.

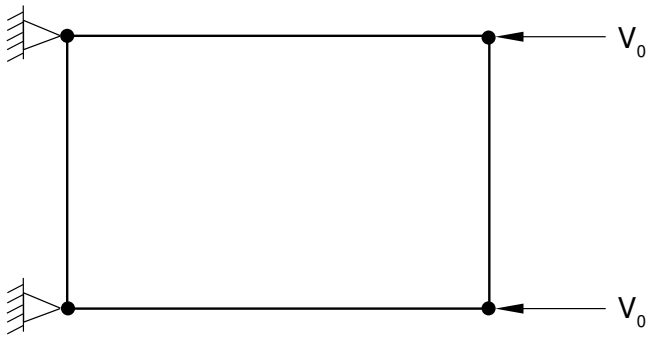


Figure 1: Element with fixed nodes and prescribed velocities.

The stable time increment size is precisely the transit time of a dilatational wave across the element. Therefore, if the initial nodal velocity is equal to the dilatational wave speed of the material, the element collapses to zero volume in one time increment. The quadratic bulk viscosity pressure introduces a resisting pressure that prevents the element from collapsing.

Fraction of critical damping due to bulk viscosity

The bulk viscosity pressures are based on only the dilatational modes of each element. The fraction of critical damping in the highest element mode is given by the following equation:

$$\xi = b_1 - b_2^2 \frac{L^e}{c_d} min\left(0, \dot{\varepsilon}_{vol}\right),$$

where ξ is the fraction of critical damping. The linear term alone represents 6% of critical damping, whereas the quadratic term is usually much smaller.

Viscous pressure

Viscous pressure loads are commonly used in structural problems and quasi-static problems to damp out the low-frequency dynamic effects, thus allowing static equilibrium to be reached in a minimal number of increments. These loads are applied as distributed loads defined by the following formula:

$$p = -c_v \left(\overline{\mathbf{v}} \cdot \overline{\mathbf{n}} \right),$$

where p is the pressure applied to the body; c_v is the coefficient of viscosity, given on the data line as the magnitude of the load; $\overline{\mathbf{v}}$ is the velocity vector of the point on the surface where the viscous pressure is being applied; and $\overline{\mathbf{n}}$ is the unit outward normal vector to the surface at the same point. For typical structural problems it is not desirable to absorb all of the energy. Typically, c_v is set equal to a small percentage—perhaps 1 or 2 percent—of the quantity ρc_d as an effective way of minimizing ongoing dynamic effects.

Material damping

The material model itself may provide damping in the form of plastic dissipation or viscoelasticity. For many applications such damping may be adequate. Another option is to use Rayleigh damping. There are two damping factors associated with Rayleigh damping: α_R for mass proportional damping and β_R for stiffness proportional damping.

Mass proportional damping

The α_R factor defines a damping contribution proportional to the mass matrix for an element. The damping forces that are introduced are caused by the absolute velocities of nodes in the model. The resulting effect can be likened to the model moving through a viscous fluid so that any motion of any point in the model triggers damping forces. Reasonable mass proportional damping does not reduce the stability limit significantly.

Stiffness proportional damping

The β_R factor defines damping proportional to the elastic material stiffness. A "damping stress,"σ_d, proportional to the total strain rate is introduced, using the following formula:

$$\tilde{\sigma}_d = \beta_R \widetilde{D}^{el} \dot{\varepsilon},$$

where $\dot{\varepsilon}$ is the strain rate. For hyperelastic and hyperfoam materials $\widetilde{D}^{el}$ is defined as the initial elastic stiffness. For all other materials $\widetilde{D}^{el}$ is the material's current elastic stiffness. This damping stress is added to the stress caused by the constitutive response at the integration point when the dynamic equilibrium equations are formed, but it is not included in the stress output. Damping can be introduced for any nonlinear analysis and provides standard Rayleigh damping for linear analyses. For a linear analysis stiffness proportional damping is exactly the same as defining a damping matrix equal to β_R times the stiffness matrix. Stiffness proportional damping must be used with caution because it may significantly reduce the stability limit. To avoid a dramatic drop in the stable time increment, the stiffness proportional damping factor, β_R, should be less than or of the same order of magnitude as the initial stable time increment without damping.

Discrete dashpots

Yet another option is to define individual dashpot elements. Each dashpot element provides a damping force proportional to the relative velocity of its two nodes. The advantage of this approach is that it enables you to apply damping only at points where you decide it is necessary. Dashpots always should be used in parallel with other elements, such as springs or trusses, so that they do not cause a significant reduction in the stability limit.

Energy balance

Energy output is often an important part of an Abaqus/Explicit analysis. Comparisons between various energy components can be used to help evaluate whether an analysis is yielding an appropriate response.

In this section:

- *Statement of energy balance*
- *Output of the energy balance*

Statement of energy balance

An energy balance for the entire model can be written as

$$E_I + E_V + E_{FD} + E_{KE} + E_{IHE} - E_W - E_{PW} - E_{CW} - E_{MW} - E_{HF} = E_{total} = \text{constant},$$

where E_I is the internal energy, E_V is the viscous energy dissipated, E_{FD} is the frictional energy dissipated, E_{KE} is the kinetic energy, E_{IHE} is the internal heat energy, E_W is the work done by the externally applied loads, and E_{PW}, E_{CW}, and E_{MW} are the work done by contact penalties, by constraint penalties, and by propelling added mass, respectively. E_{HF} is the external heat energy through external fluxes. The sum of these energy components is E_{total}, which should be constant. In the numerical model E_{total} is only approximately constant, generally with an error of less than 1%.

Internal energy

The internal energy is the sum of the recoverable elastic strain energy, E_E; the energy dissipated through inelastic processes such as plasticity, E_P; the energy dissipated through viscoelasticity or creep, E_{CD}; the artificial strain energy, E_A; the energy dissipated through damage, E_{DMD}; the energy dissipated through distortion control, E_{DC}; and the fluid cavity energy, E_{FC}:

$$E_I = E_E + E_P + E_{CD} + E_A + E_{DMD} + E_{DC} + E_{FC}.$$

The artificial strain energy includes energy stored in hourglass resistances and transverse shear in shell and beam elements. Large values of artificial strain energy indicate that mesh refinement or other changes to the mesh are necessary.

Viscous energy

The viscous energy is the energy dissipated by damping mechanisms, including bulk viscosity damping and material damping. A fundamental variable in the global energy balance, viscous energy is not part of the energy dissipated through viscoelasticity or inelastic processes.

External work of applied forces

The external work is integrated forward continuously, defined entirely by nodal forces (moments) and displacements (rotations). Prescribed boundary conditions also contribute to the external work.

Output of the energy balance

Each of the energy quantities can be requested as output and can be plotted as time histories summed over the entire model, particular element sets, individual elements, or as energy density within each element. The variable names associated with the energy quantities summed over the entire model or element sets are as listed in *Table 1*.

Table 1: Whole model energy output variables.

Variable Name	Energy Quantity
ALLIE	Internal energy, E_I: ALLIE = ALLSE + ALLPD + ALLCD + ALLAE + ALLDMD + ALLDC + ALLFC.
ALLKE	Kinetic energy, E_{KE}.
ALLVD	Viscous dissipated energy, E_V.
ALLFD	Frictional dissipated energy, E_{FD}.
ALLCD	Energy dissipated by viscoelasticity, E_{CD}.
ALLWK	Work of the external forces, E_W.
ALLPW	Work done by contact penalties, E_{PW}.
ALLCW	Work done by constraint penalties, E_{CW}.
ALLMW	Work done by propelling added mass (due to mass scaling), E_{MW}.
ALLSE	Elastic strain energy, E_E.
ALLPD	Inelastic dissipated energy, E_P.
ALLAE	Artificial strain energy, E_A.
ALLIHE	Internal heat energy, E_{IHE}.
ALLHF	External heat energy through external fluxes, E_{HF}.
ALLDMD	Energy dissipated by damage, E_{DMD}.
ALLDC	Energy dissipated by distortion control, E_{DC}.
ALLFC	Fluid cavity energy (negative of work done by fluid cavities), E_{FC}.
ETOTAL	Energy balance: $E_{TOT} = E_I + E_V + E_{FD} + E_{KE} + E_{IHE} - E_W - E_{PW} - E_{CW} - E_{MW} - E_{HF}$.

In addition, Abaqus/Explicit can produce element-level energy output and energy density output, as listed in *Table 2*.

Table 2: Whole element energy output variables.

Variable Name	Whole Element Energy Quantity
ELSE	Elastic strain energy.
ELPD	Plastic dissipated energy.
ELCD	Creep dissipated energy.

Variable Name	Whole Element Energy Quantity
ELVD	Viscous dissipated energy.
ELASE	Artificial energy = drill energy + hourglass energy.
EKEDEN	Kinetic energy density in the element.
ESEDEN	Elastic strain energy density in the element.
EPDDEN	Plastic energy density dissipated in the element.
EASEDEN	Artificial strain energy density in the element.
ECDDEN	Creep strain energy density dissipated in the element.
EVDDEN	Viscous energy density dissipated in the element.
ELDMD	Energy dissipated in the element by damage.

Summary

This section presents a summary of the concepts.

- Abaqus/Explicit uses a central difference rule to integrate the kinematics explicitly through time.
- The explicit method requires many small time increments. Since there are no simultaneous equations to solve, each increment is inexpensive.
- The explicit method has great cost savings over the implicit method as the model size increases.
- The *stability limit* is the maximum time increment that can be used to advance the kinematic state and still remain accurate.
- Abaqus/Explicit automatically controls the time increment size throughout the analysis to maintain stability.
- As the material stiffness increases, the stability limit decreases; as the material density increases, the stability limit increases.
- For a mesh with a single material, the stability limit is roughly proportional to the smallest element dimension.
- Generally, mass proportional damping is used in Abaqus/Explicit to damp low-frequency oscillations, and stiffness proportional damping is used to damp high-frequency oscillations.

Materials

The material library in Abaqus allows most engineering materials to be modeled, including metals, plastics, rubbers, foams, composites, granular soils, rocks, and plain and reinforced concrete. This guide discusses only three of the most commonly used material models: linear elasticity, metal plasticity, and rubber elasticity. All of the material models are discussed in detail in *Abaqus Materials Guide*.

In this section:

- *Defining materials in Abaqus*
- *Plasticity in ductile metals*
- *Selecting elements for elastic-plastic problems*
- *Example: connecting lug with plasticity*
- *Example: blast loading on a stiffened plate*
- *Hyperelasticity*
- *Example: axisymmetric mount*
- *Mesh design for large distortions*
- *Techniques for reducing volumetric locking*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Defining materials in Abaqus

You can use any number of different materials in your simulation. Each material definition is given a name. Different regions in a model are associated with different material definitions through the assignment of section properties that refer to the material name.

Plasticity in ductile metals

This section discusses the definition of plasticity in ductile metals.

In this section:

- *Characteristics of plasticity in ductile metals*
- *Stress and strain measures for finite deformations*
- *Defining plasticity in Abaqus*

Characteristics of plasticity in ductile metals

Many metals have approximately linear elastic behavior at low strain magnitudes, and the stiffness of the material, known as the Young's or elastic modulus, is constant.

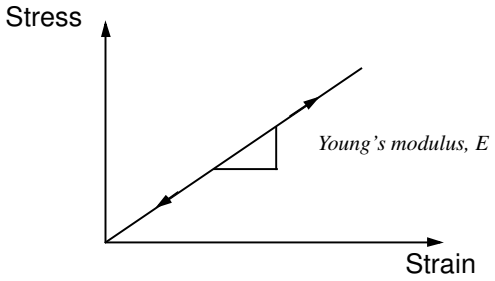


Figure 1: Stress-strain behavior for a linear elastic material, such as steel, at small strains.

At higher stress (and strain) magnitudes, metals begin to have nonlinear, inelastic behavior (see *Figure 2*), which is referred to as plasticity.

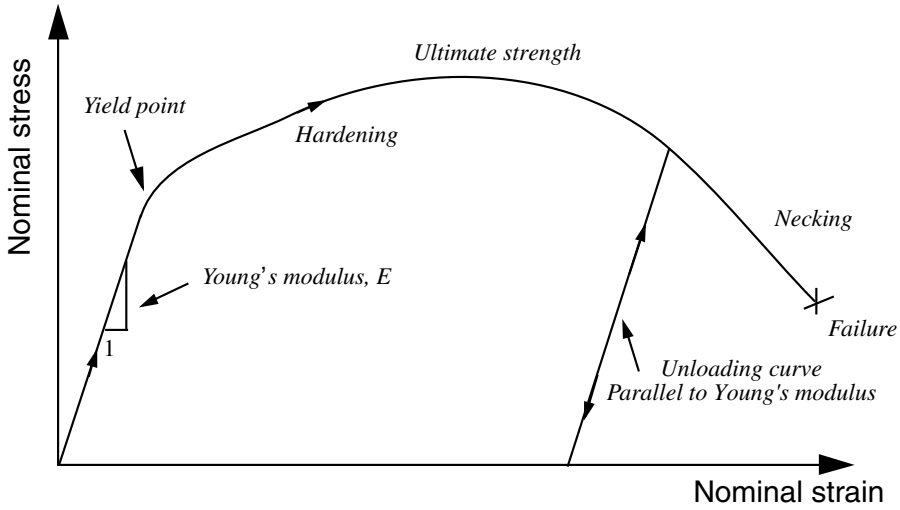


Figure 2: Nominal stress-strain behavior of an elastic-plastic material in a tensile test.

The plastic behavior of a material is described by its yield point and its post-yield hardening. The shift from elastic to plastic behavior occurs at a certain point, known as the elastic limit or yield point, on a material's stress-strain curve (see *Figure 2*). The stress at the yield point is called the yield stress. In most metals the initial yield stress is 0.05 to 0.1% of the material's elastic modulus.

The deformation of the metal prior to reaching the yield point creates only elastic strains, which are fully recovered if the applied load is removed. However, once the stress in the metal exceeds the yield stress, permanent (inelastic) deformation begins to occur. The strains associated with this permanent deformation are called plastic strains. Both elastic and plastic strains accumulate as the metal deforms in the post-yield region.

The stiffness of a metal typically decreases dramatically once the material yields (see *Figure 2*). A ductile metal that has yielded will recover its initial elastic stiffness when the applied load is removed (see *Figure 2*). Often the plastic deformation of the material increases its yield stress for subsequent loadings: this behavior is called work hardening.

Another important feature of metal plasticity is that the inelastic deformation is associated with nearly incompressible material behavior. Modeling this effect places some severe restrictions on the type of elements that can be used in elastic-plastic simulations.

A metal deforming plastically under a tensile load may experience highly localized extension and thinning, called *necking*, as the material fails (see *Figure 2*). The engineering stress (force per unit undeformed area) in the metal is known as the *nominal stress*, with the conjugate *nominal strain* (length change per unit undeformed length). The nominal stress in the metal as it is necking is much lower than the material's ultimate strength. This material behavior is caused by the geometry of the test specimen, the nature of the test itself, and the stress and strain measures used. For example, testing the same material in compression produces a stress-strain plot that does not have a necking region because the specimen is not going to thin as it deforms under compressive loads. A mathematical model describing the plastic behavior of metals should be able to account for differences in the compressive and tensile behavior independent of the structure's geometry or the nature of the applied loads. This goal can be accomplished if the familiar definitions of nominal stress, F/A_0, and nominal strain, $\Delta l/l_0$, where the subscript 0 indicates a value from the undeformed state of the material, are replaced by new measures of stress and strain that account for the change in area during the finite deformations.

Stress and strain measures for finite deformations

Strains in compression and tension are the same only if considered in the limit as $\Delta l \to dl \to 0$; i.e.,

$$d\varepsilon = \frac{dl}{l}$$

and

$$\varepsilon = \int_{l_0}^{l} \frac{dl}{l} = \ln\left(\frac{l}{l_0}\right),$$

where l is the current length, l_0 is the original length, and ε is the *true strain* or *logarithmic strain*.

The stress measure that is the conjugate to the true strain is called the *true stress* and is defined as

$$\sigma = \frac{F}{A},$$

where F is the force in the material and A is the current area. A ductile metal subjected to finite deformations will have the same stress-strain behavior in tension and compression if true stress is plotted against true strain.

Defining plasticity in Abaqus

When defining plasticity data in Abaqus, you must use *true stress* and *true strain*. Abaqus requires these values to interpret the data correctly.

Quite often material test data are supplied using values of nominal stress and strain. In such situations you must use the expressions presented below to convert the plastic material data from nominal stress-strain values to true stress-strain values.

The relationship between true strain and nominal strain is established by expressing the nominal strain as

$$\varepsilon_{nom} = \frac{l - l_0}{l_0} = \frac{l}{l_0} - \frac{l_0}{l_0} = \frac{l}{l_0} - 1.$$

Adding unity to both sides of this expression and taking the natural log of both sides provides the relationship between the true strain and the nominal strain:

$$\varepsilon = \ln(1 + \varepsilon_{nom}).$$

The relationship between true stress and nominal stress is formed by considering the incompressible nature of the plastic deformation and assuming the elasticity is also incompressible, so

$$l_0 A_0 = lA.$$

The current area is related to the original area by

$$A = A_0 \frac{l_0}{l}.$$

Substituting this definition of A into the definition of true stress gives

$$\sigma = \frac{F}{A} = \frac{F}{A_0}\frac{l}{l_0} = \sigma_{nom}\left(\frac{l}{l_0}\right),$$

where

$$\frac{l}{l_0}$$

can also be written as

$$1 + \varepsilon_{nom}.$$

Making this final substitution provides the relationship between true stress and nominal stress and strain:

$$\sigma = \sigma_{nom}\left(1 + \varepsilon_{nom}\right).$$

These relationships are valid only prior to necking.

The classical metal plasticity model in Abaqus defines the post-yield behavior for most metals. Abaqus approximates the smooth stress-strain behavior of the material with a series of straight lines joining the given data points. Any number of points can be used to approximate the actual material behavior; therefore, it is possible to use a very close approximation of the actual material behavior. The plastic data define the true yield stress of the material as a function of true plastic strain. The first piece of data given defines the initial yield stress of the material and, therefore, should have a plastic strain value of zero.

The strains provided in material test data used to define the plastic behavior are not likely to be the plastic strains in the material. Instead, they will probably be the total strains in the material. You must decompose these total strain values into the elastic and plastic strain components. The plastic strain is obtained by subtracting the elastic strain, defined as the value of true stress divided by the Young's modulus, from the value of total strain (see *Figure 1*).

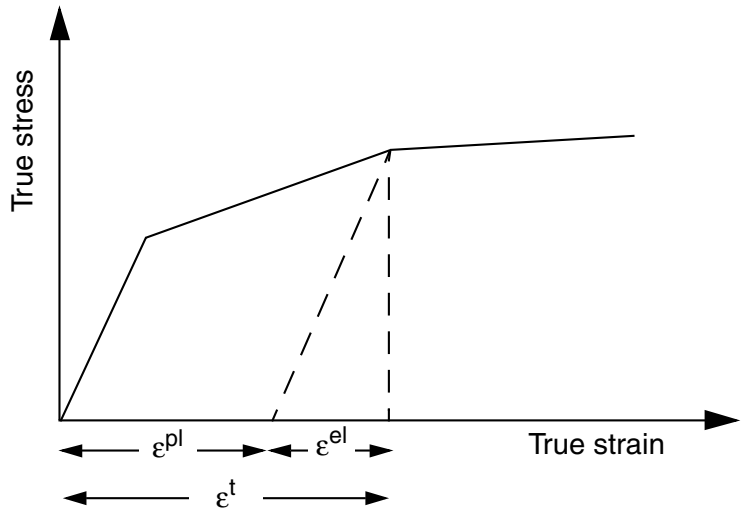


Figure 1: Decomposition of the total strain into elastic and plastic components.

This relationship is written

$$\varepsilon^{pl} = \varepsilon^{t} - \varepsilon^{el} = \varepsilon^{t} - \sigma/E,$$

where

ε^{pl}

is true plastic strain,

ε^{t}

is true total strain,

ε^{el}

is true elastic strain,

σ

is true stress, and

E

is Young's modulus.

Example of converting material test data to Abaqus input

The nominal stress-strain curve in *Figure 2* will be used as an example of how to convert the test data defining a material's plastic behavior into the appropriate input format for Abaqus. The six points shown on the nominal stress-strain curve will be used to determine the plastic data.

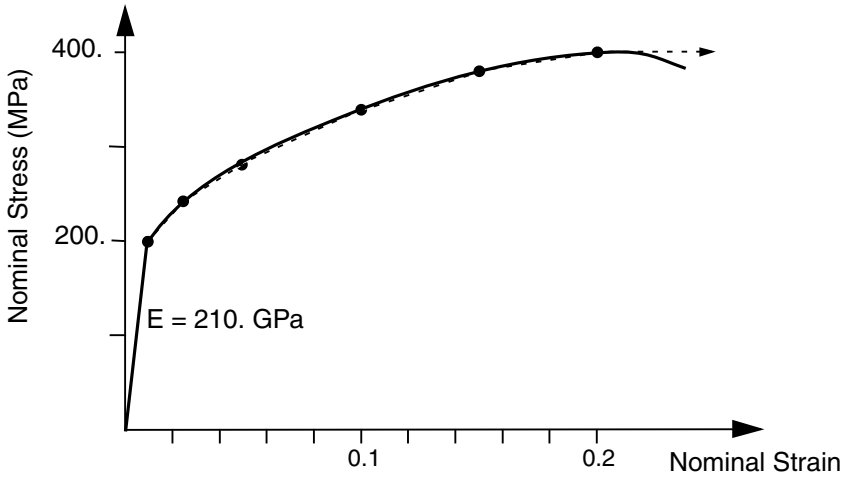


Figure 2: Elastic-plastic material behavior.

The first step is to use the equations relating the true stress to the nominal stress and strain and the true strain to the nominal strain (shown earlier) to convert the nominal stress and nominal strain to true stress and true strain. Once these values are known, the equation relating the plastic strain to the total and elastic strains (shown earlier) can be used to determine the plastic strains associated with each yield stress value. The converted data are shown in *Table 1*.

Table 1: Stress and strain conversions.

Nominal Stress (Pa)	Nominal Strain	True Stress (Pa)	True Strain	Plastic Strain
200E6	0.00095	200.2E6	0.00095	0.0
240E6	0.025	246E6	0.0247	0.0235
280E6	0.050	294E6	0.0488	0.0474
340E6	0.100	374E6	0.0953	0.0935
380E6	0.150	437E6	0.1398	0.1377
400E6	0.200	480E6	0.1823	0.1800

While there are few differences between the nominal and true values at small strains, there are very significant differences at larger strain values; therefore, it is extremely important to provide the proper stress-strain data to Abaqus if the strains in the simulation will be large.

Data regularization in Abaqus/Explicit

When performing an analysis, Abaqus/Explicit may not use the material data exactly as defined by the user; for efficiency, all material data that are defined in tabular form are automatically *regularized*. Material data can be functions of temperature, external fields, and internal state variables, such as plastic strain. For each material point calculation, the state of the material must be determined by interpolation, and, for efficiency, Abaqus/Explicit fits the user-defined curves with curves composed of equally spaced points. These regularized material curves are the material data used during the analysis. It is important to understand the differences that might exist between the regularized material curves used in the analysis and the curves that you specified.

To illustrate the implications of using regularized material data, consider the following two cases. *Figure 3* shows a case in which the user has defined data that are not regular.

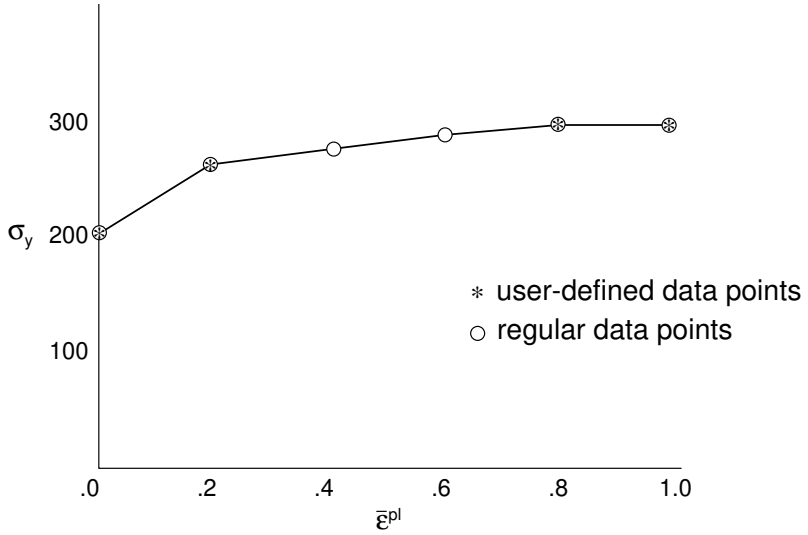


Figure 3: Example of user data that can be regularized exactly.

In this example Abaqus/Explicit generates the six regular data points shown, and the user's data are reproduced exactly. *Figure 4* shows a case where the user has defined data that are difficult to regularize exactly. In this example it is assumed that Abaqus/Explicit has regularized the data by dividing the range into 10 intervals that do not reproduce the user's data points exactly.

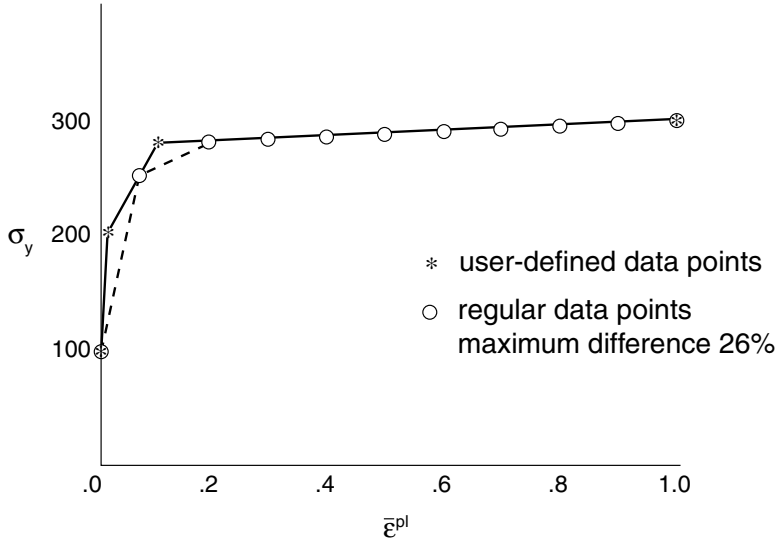


Figure 4: Example of user data that are difficult to regularize.

Abaqus/Explicit attempts to use enough intervals such that the maximum error between the regularized data and the user-defined data is less than 3%; however, you can change this error tolerance. If more than 200 intervals are required to obtain an acceptable regularized curve, the analysis stops during the data checking with an error message. In general, the regularization is more difficult if the smallest interval defined by the user is small compared to the range of the independent variable. In *Figure 4* the data point for a strain of 1.0 makes the range of strain values large compared to the small intervals defined at low strain levels. Removing this last data point enables the data to be regularized much more easily.

Interpolation between data points

Abaqus interpolates linearly between the data points provided (or, in Abaqus/Explicit, regularized data) to obtain the material's response and assumes that the response is constant outside the range defined by the input data, as shown in *Figure 5*. Thus, the stress in this material will never exceed 480 MPa; when the stress in the material reaches 480 MPa, the material will deform continuously until the stress is reduced below this value.

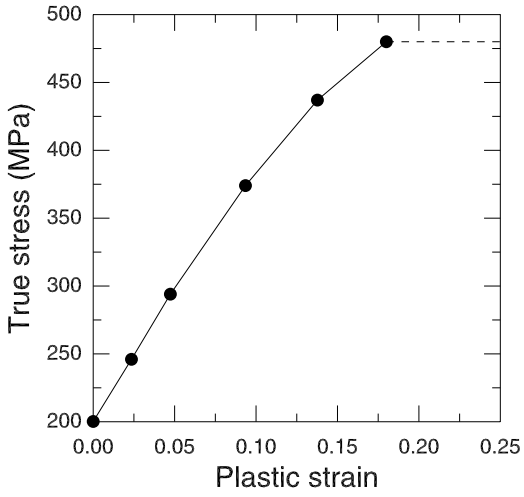


Figure 5: Material curve used by Abaqus.

Material calibration in Abaqus/CAE

Abaqus/CAE allows you to calibrate a material model from test data. With this capability, you can import material test data into Abaqus/CAE, process the data, and derive elastic and plastic isotropic material behaviors from the data. This feature is discussed further in *Creating material calibrations*.

Selecting elements for elastic-plastic problems

The incompressible nature of plastic deformation in metals places limitations on the types of elements that can be used for an elastic-plastic simulation. The limitations arise because modeling incompressible material behavior adds kinematic constraints to an element; in this case the limitations constrain the volume at the element's integration points to remain constant. In certain classes of elements the addition of these incompressibility constraints makes the element overconstrained. When these elements cannot resolve all of these constraints, they suffer from *volumetric locking*, which causes their response to be too stiff. Volumetric locking is indicated by a rapid variation of hydrostatic pressure stress from element to element or integration point to integration point.

The fully integrated, second-order, solid elements available in Abaqus/Standard are very susceptible to volumetric locking when modeling incompressible material behavior and, therefore, should not be used in elastic-plastic simulations. The fully integrated, first-order, solid elements in Abaqus/Standard do not suffer from volumetric locking because Abaqus actually uses a constant volume strain in these elements. Thus, they can be used safely in plasticity problems.

Reduced-integration solid elements have fewer integration points at which the incompressibility constraints must be satisfied. Therefore, they are not overconstrained and can be used for most elastic-plastic simulations. The second-order reduced-integration elements in Abaqus/Standard should be used with caution if the strains exceed 20–40% because at this magnitude they can suffer from volumetric locking. This effect can be reduced with mesh refinement.

If you have to use fully integrated, second-order elements in Abaqus/Standard, use the hybrid versions, which are designed to model incompressible behavior; however, the additional degrees of freedom in these elements will make the analysis more computationally expensive.

A family of modified second-order triangular and tetrahedral elements is available that provides improved performance over the first-order triangular and tetrahedral elements and that avoids some of the problems that exist for conventional second-order triangular and tetrahedral elements. In particular, these elements exhibit minimal shear and volumetric locking. These elements are available in addition to fully integrated and hybrid elements in Abaqus/Standard; they are the only second-order continuum (solid) elements available in Abaqus/Explicit.

Example: connecting lug with plasticity

In this example you will investigate what happens if a steel connecting lug is subjected to an extreme load.

In this section:

- *Connecting lug with plasticity analysis*
- *Modifications to the model*
- *Job monitor and diagnostics*
- *Postprocessing the results*
- *Adding hardening to the material model*
- *Running the analysis with plastic hardening*
- *Postprocessing the results*

Connecting lug with plasticity analysis

You will investigate what happens if the steel connecting lug from a previous example is subjected to an extreme load (60 kN) caused by an accident.
The steel connecting lug was modeled in *Example: connecting lug*.

The results from the linear analysis indicate that the lug will yield. You need to determine the extent of the plastic deformation in the lug and the magnitude of the plastic strains so that you can assess whether or not the lug will fail. You do not need to consider inertial effects in this analysis; thus, you will use Abaqus/Standard to examine the static response of the lug.

The only inelastic material data available for the steel are its yield stress (380 MPa) and its strain at failure (0.15). You decide to assume that the steel is perfectly plastic: the material does not harden, and the stress can never exceed 380 MPa (see *Figure 1*).

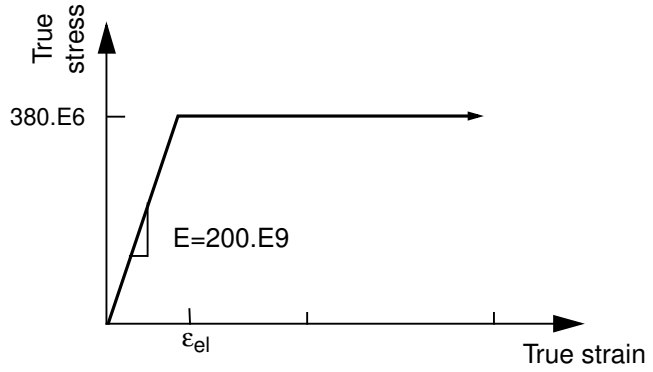


Figure 1: Stress-strain behavior for the steel.

In reality some hardening will probably occur, but this assumption is conservative; if the material hardens, the plastic strains will be less than those predicted by the simulation.

Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Connecting lug with plasticity*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Connecting lug with plasticity**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Modifications to the model

Open the model database file `Lug.cae`, and copy the model `Elastic` to a model named `Plastic`.

Material definition

For the `Plastic` model, you will specify the post-yield behavior of the material using the classical metal plasticity model in Abaqus. The initial yield stress at zero plastic strain is 380 MPa. Since you are modeling the steel as perfectly plastic, no other yield stresses are required. You will perform a general, nonlinear simulation because of the nonlinear material behavior in the model.

To add plasticity data to the material model:

1. In the Model Tree, expand the **Materials** container and double-click **Steel**.
2. In the material editor, select **Mechanical->Plasticity->Plastic** to invoke the classical metal plasticity model. Enter an initial yield stress of `380.E6` with a corresponding initial plastic strain of `0.0`.

Step definition and output requests

Edit the step definition and output requests. In the **Basic** tabbed page of the **Edit Step** dialog box, accept the total time period of `1.0` and assume that the effects of geometric nonlinearity will not be important in this simulation. In the **Incrementation** tabbed page, specify an initial increment size that is 20% of the total step time (`0.2`). This simulation is a static analysis of the lug under extreme loads; you do not know in advance how many increments this simulation may require. The default maximum of `100` increments, however, is reasonably large and should be sufficient for this analysis.

Open the **Field Output Requests Manager**. Edit the current output request to request the preselected defaults at every increment.

Loading

The load applied in this simulation is twice what was applied in the linear elastic simulation of the lug (60 kN vs. 30 kN). Therefore, in the Model Tree, double-click **Pressure load** underneath the **Loads** container, and double the magnitude of the pressure applied to the lug (i.e., change the magnitude to `10.0E7`).

Job definition

In the Model Tree, create a job named `PlasticLugNoHard`, and enter the following job description: `Elastic-Plastic Steel Connecting Lug`. Remember to save your model database file.

Submit the job for analysis, and monitor the solution progress. Correct any modeling errors, and investigate the source of any warning messages. This analysis should terminate prematurely; the reasons are discussed in the following section.

Job monitor and diagnostics

You can monitor the progress of your analysis while it is running by looking at the **Job Monitor**.

Job Monitor

When Abaqus/Standard has finished the simulation, the **Job Monitor** will contain information similar to that shown in *Figure 1*.

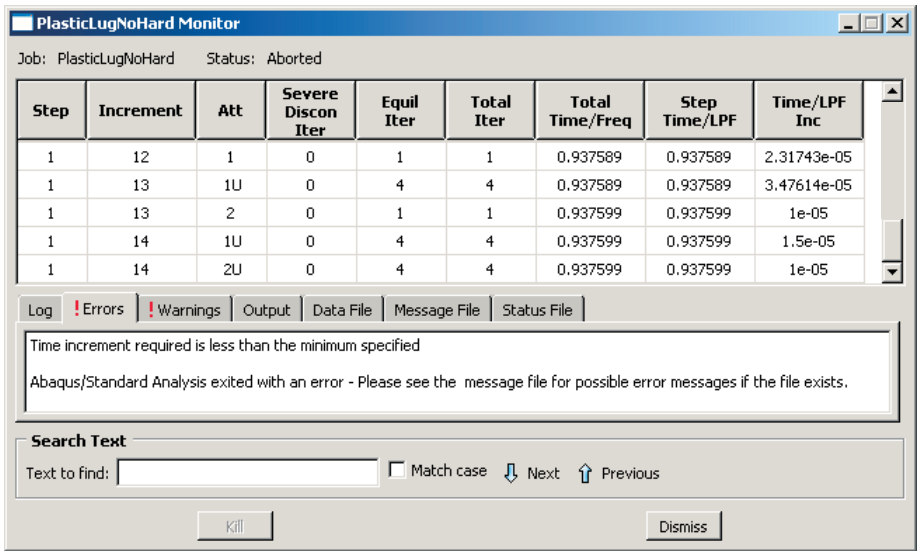


Figure 1: Job Monitor: perfectly plastic connecting lug.

Abaqus/Standard was able to apply only 94% of the prescribed load to the model. The **Job Monitor** shows that Abaqus/Standard reduced the size of the time increment, shown in the last (right-hand) column, many times during the simulation and stopped the analysis in the fourteenth increment. The information on the **Errors** tabbed page (see *Figure 1*) indicates that the analysis terminated. Click the **Message File** tab to view the error details in the message file, as shown in *Figure 2*. The error indicates that the analysis terminated because the size of the time increment is smaller than the value allowed for this analysis. This is a classic symptom of convergence difficulties and is a direct result of the continued reduction in the time increment size. To begin diagnosing the problem, click the **Warnings** tab in the **Job Monitor** dialog box. As shown in *Figure 3*, many warning messages concerning large strain increments and problems with the plasticity calculations are found here. These warnings are related since problems with the plasticity calculations are typically the result of excessively large strain increments and often lead to divergence. Thus, we suspect that numerical problems with the plasticity calculations caused Abaqus/Standard to terminate the analysis early.

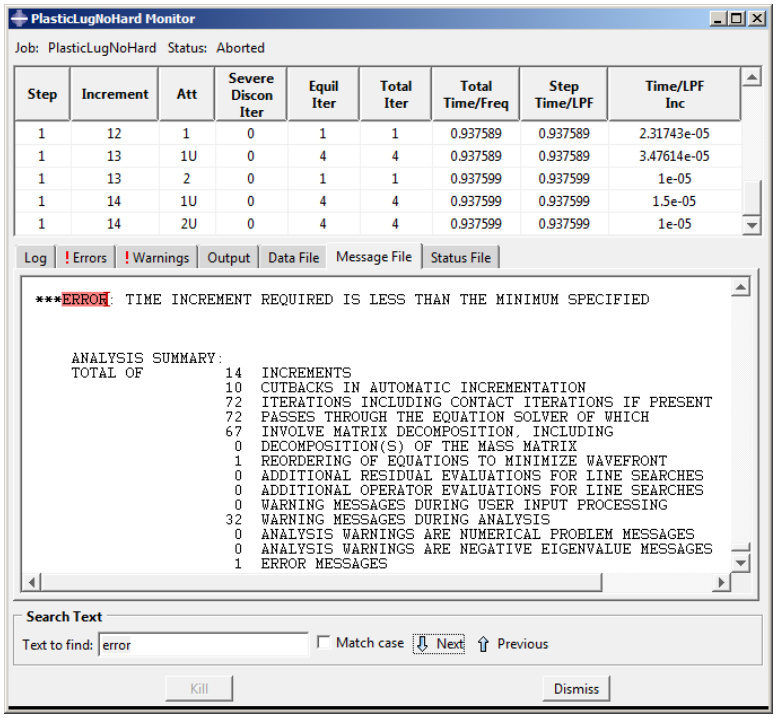


Figure 2: Message File: error description.

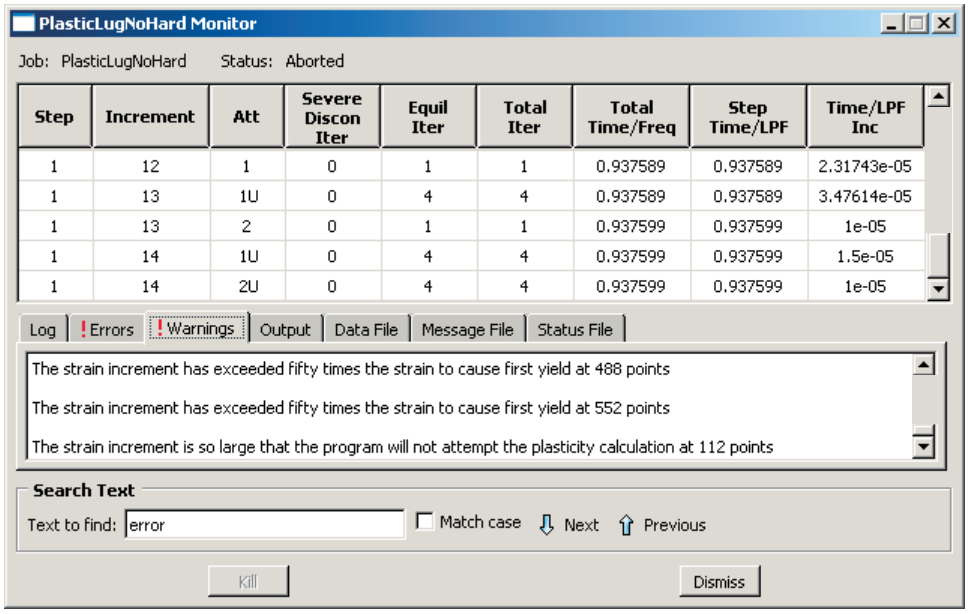


Figure 3: Warnings: perfectly plastic connecting lug.

Job diagnostics

Enter the Visualization module, and open the file `PlasticLugNoHard.odb`. Open the **Job Diagnostics** dialog box to examine the convergence history of the job. Looking at the information for the first increment in the analysis (see *Figure 4*), you will discover that the model's initial behavior is determined to be linear.

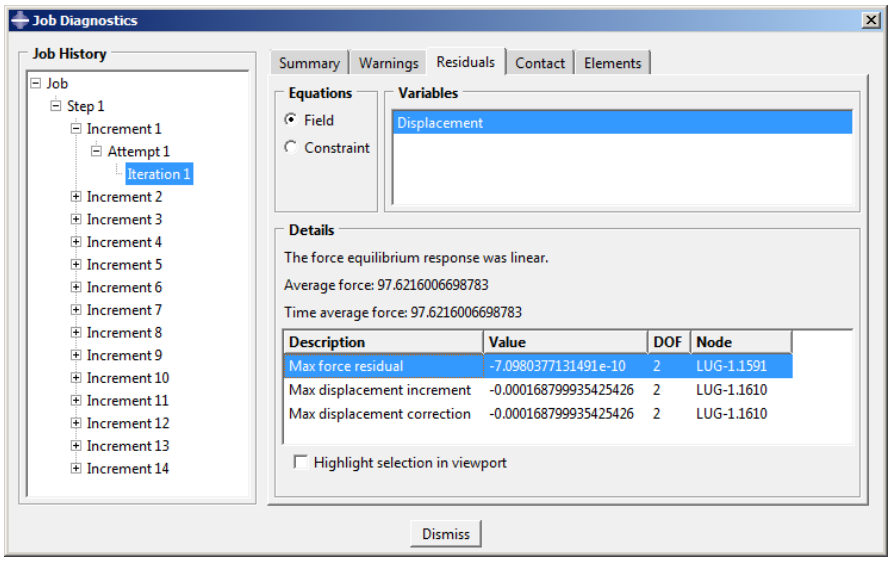


Figure 4: Convergence history for Increment 1.

This judgement is based on the fact that the magnitude of the residual, r^{α}_{max}, is less than $10^{-8}\bar{q}^{\alpha}$ (the time average force); the displacement correction criterion is ignored in this case. The model's behavior is also linear in the second increment (see *Figure 5*).

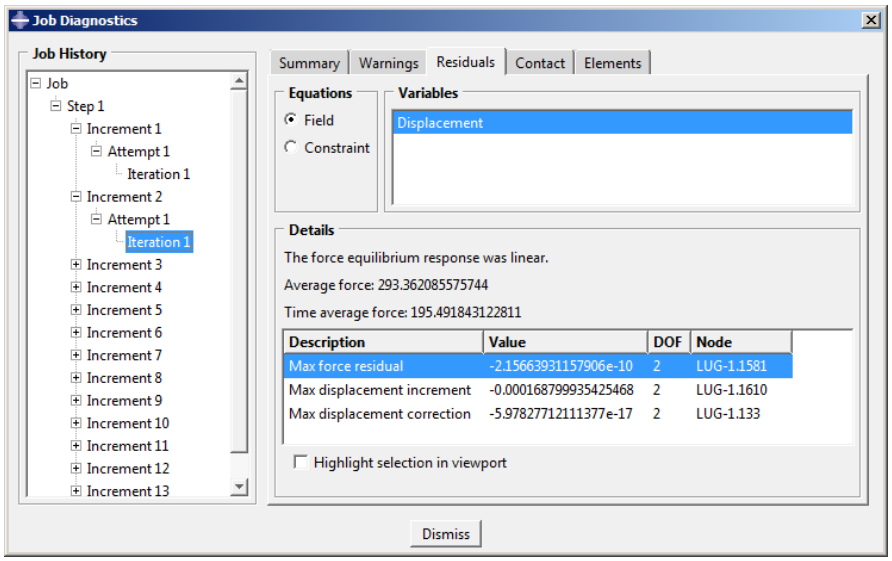


Figure 5: Convergence history for Increment 2.

Abaqus/Standard requires several iterations to obtain a converged solution in the third increment, which indicates that nonlinear behavior occurs in the model during this increment. The only nonlinearity in the model is the plastic material behavior, so the steel must have started to yield somewhere in the lug at this applied load magnitude. The summary of the final (converged) iteration for the third increment is shown in *Figure 6*.

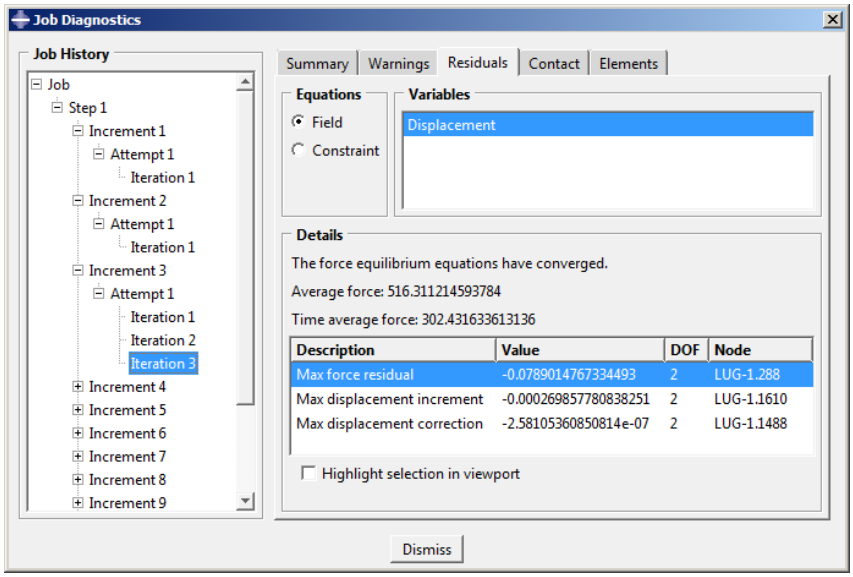


Figure 6: Convergence history for Increment 3.

Abaqus/Standard attempts to find a solution in the fourth increment using an increment size of 0.3, which means it is applying 30% of the total load, or 18 kN, during this increment. After several iterations, Abaqus/Standard abandons the attempt and reduces the size of the time increment to 25% of the value used in the first attempt. This reduction in increment size is called a *cutback*. With the smaller increment size, Abaqus/Standard finds a converged solution in just a few iterations.

Look more closely at the information for the first attempt of the fourth increment (this is where the convergence difficulties first appear). For this attempt Abaqus/Standard detects large strain increments at the integration points of a number of elements, as shown in *Figure 7*.

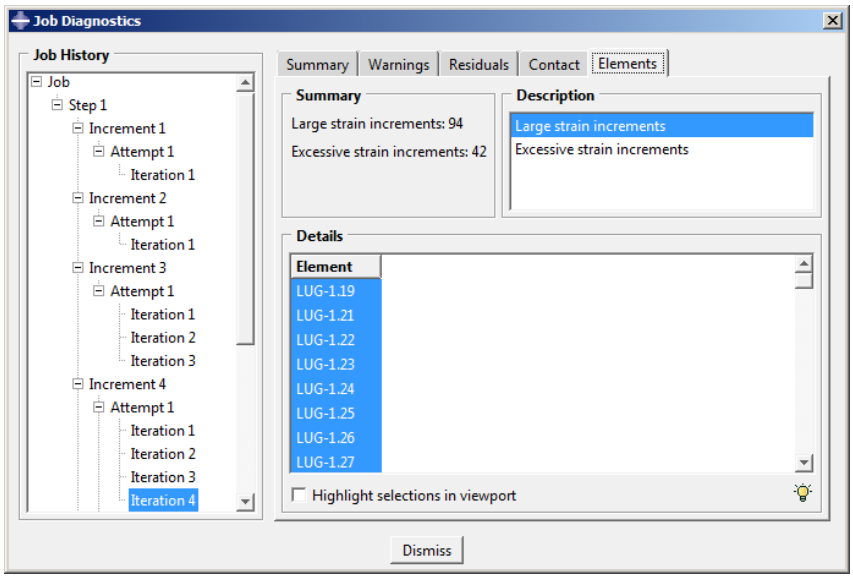


Figure 7: Convergence history for Increment 4.

"Large" strain increments are those that exceed the strain at initial yield by 50 times; some of these increments are also considered "excessive," which implies the plasticity calculations are not even attempted at the affected integration points. Thus, we see that the onset of the convergence difficulties is directly related to the large strain increments and problems with the plasticity calculations.

Abaqus/Standard encounters renewed convergence difficulties in subsequent increments until finally it terminates the job. In many of these increments Abaqus/Standard cuts back the time increment size because the strain increments are so large that the plasticity calculations are not even performed. Thus, we conclude the overall convergence difficulties are indeed the result of numerical problems with the plasticity calculations.

This check on the magnitude of the total strain increment is an example of the many automatic solution controls Abaqus/Standard uses to ensure that the solution obtained for your simulation is both accurate and efficient. The automatic solution controls are suitable for almost all simulations. Therefore, you do not have to worry about providing parameters to control the solution algorithm: you only have to be concerned with the input data for your model.

An interesting observation is made using the **Job Diagnostics** dialog box: in virtually all attempts where convergence problems are encountered, the elements with large or excessive strain increments are in the vicinity of the built-in end of the lug (where yielding begins) while the node with the largest displacement correction is in the vicinity of the loaded end of the lug. This implies that the loaded end wants to deform more than the built-in end can support. Deformed model shape plots can help you pursue this observation further.

Postprocessing the results

Look at the results in the Visualization module to understand what caused the excessive plasticity.

Plotting the deformed model shape

Create a plot of the model's deformed shape, and check that this shape is realistic.

The default view is isometric. You can set the view shown in *Figure 1* by using the options in the **View** menu or the tools in the View Manipulation toolbar; in this figure perspective is also turned off.

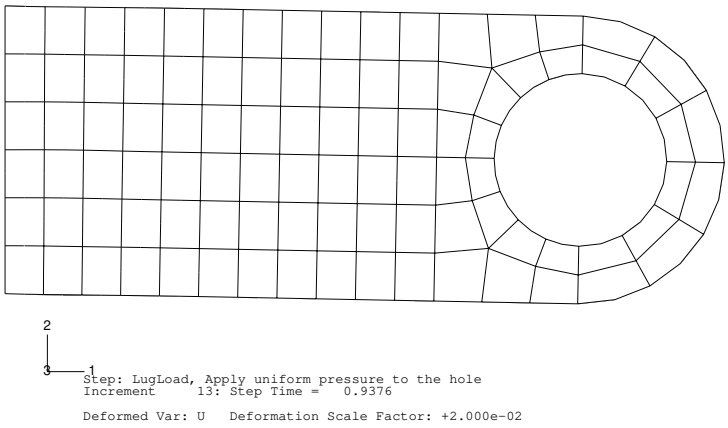


Figure 1: Deformed model shape using results for the simulation without hardening.

The displacements and, particularly, the rotations of the lug shown in the plot are large but do not seem large enough to have caused all of the numerical problems seen in the simulation. Look closely at the information in the plot's title for an explanation. The deformation scale factor used in this plot is 0.02; i.e., the displacements are scaled to 2% of their actual values. (Your deformation scale factor may be different.)

Abaqus/CAE always scales the displacements in a geometrically linear simulation such that the deformed shape of the model fits into the viewport. (This is in contrast to a geometrically nonlinear simulation, where Abaqus/CAE does not scale the displacements and, instead, adjusts the view by zooming in or out to fit the deformed shape in the plot.) To plot the actual displacements, set the deformation scale factor to 1.0. This will produce a plot of the model in which the lug has deformed until it is almost parallel to the vertical (global *Y*) axis.

The applied load of 60 kN exceeds the limit load of the lug, and the lug collapses when the material yields at all the integration points through its thickness. The lug then has no stiffness to resist further deformation because of the perfectly plastic post-yield behavior of the steel. This is consistent with the pattern observed earlier concerning the locations of the large strain increments and maximum displacement corrections.

Adding hardening to the material model

The connecting lug simulation with perfectly plastic material behavior predicts that the lug will suffer catastrophic failure caused by the collapse of the structure. We have already mentioned that the steel would probably exhibit some hardening after it has yielded. You suspect that including hardening behavior would allow the lug to withstand this 60 kN load because of the additional stiffness it would provide. Therefore, you decide to add some hardening to the steel's material property definition. Assume that the yield stress increases to 580 MPa at a plastic strain of 0.35, which represents typical hardening for this class of steel. The stress-strain curve for the modified material model is shown in *Figure 1*.

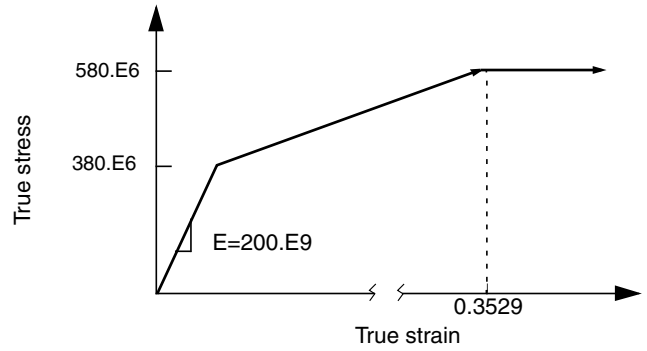


Figure 1: Modified stress-strain behavior of the steel.

Modify your plastic material data so that it includes the hardening data. Edit the material definition to add a second row of data to the plastic data form. Enter a yield stress of `580.E6` with a corresponding plastic strain of `0.35`.

Running the analysis with plastic hardening

Create a job named `PlasticLugHard`. Submit the job for analysis, and monitor the solution progress. Correct any modeling errors, and investigate the source of any warning messages.

Job Monitor

The summary of the analysis in the **Job Monitor**, shown in *Figure 1*, indicates that Abaqus/Standard found a converged solution when the full 60 kN load was applied. The hardening data added enough stiffness to the lug to prevent it from collapsing under the 60 kN load.

There are no convergence-related warnings issued during the analysis, so you can proceed directly to postprocessing the results.

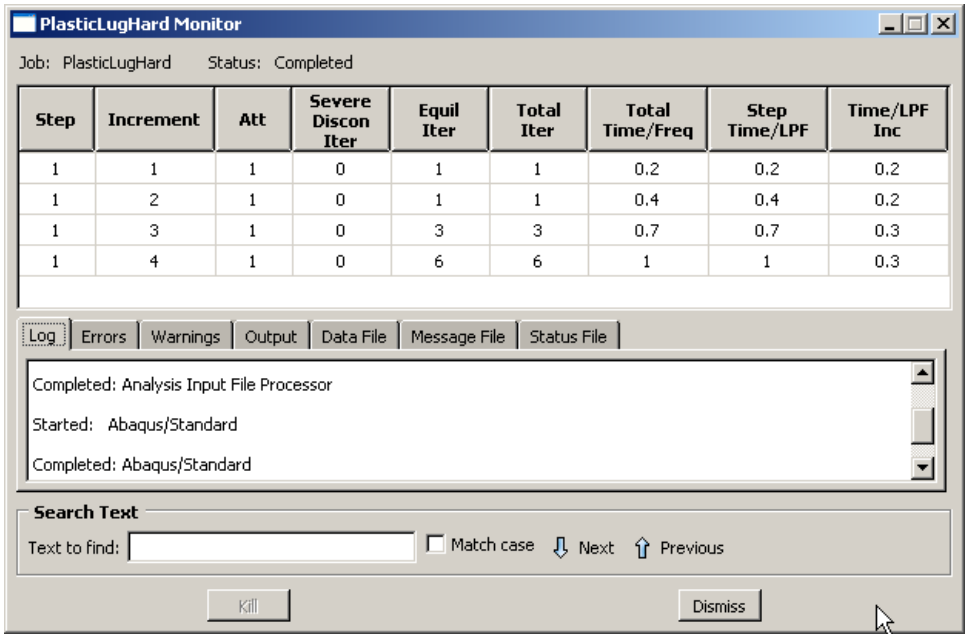


Figure 1: Job Monitor: connecting lug with plastic hardening.

Postprocessing the results

Enter the Visualization module, and open the file `PlasticLugHard.odb`.

Deformed model shape and peak displacements

Plot the deformed model shape with these new results, and change the deformation scale factor to 2 to obtain a plot similar to *Figure 1*. The displayed deformations are double the actual deformations.

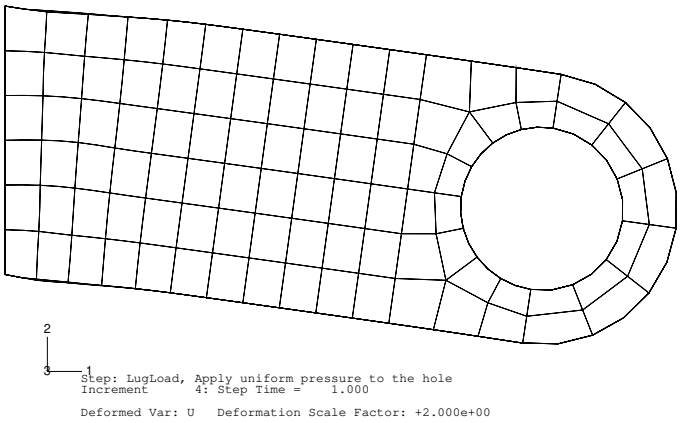


Figure 1: Deformed model shape for the simulation with plastic hardening.

Contour plot of Mises stress

Contour the Mises stress in the model. Create a filled contour plot using ten contour intervals on the actual deformed shape of the lug (i.e., set the deformation scale factor to 1.0) with the plot title and state blocks suppressed. Use the view manipulation tools to position and size the model to obtain a plot similar to that shown in *Figure 2*.

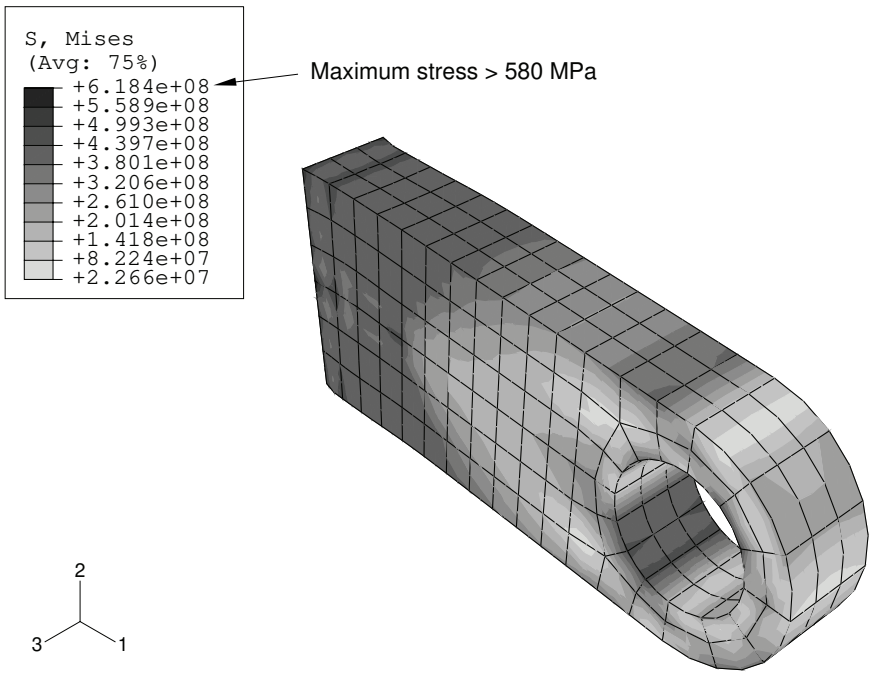


Figure 2: Contour of Mises stress.

Do the values listed in the contour legend surprise you? The maximum stress is greater than 580 MPa, which should not be possible since the material was assumed to be perfectly plastic at this stress magnitude. This misleading result occurs because of the algorithm that Abaqus/CAE uses to create contour plots for element variables, such as stress. The contouring algorithm requires data at the nodes; however, Abaqus/Standard calculates element variables at the integration points. Abaqus/CAE calculates nodal values of element variables by extrapolating the data from the integration points to the nodes. The extrapolation order depends on the element type; for second-order, reduced-integration elements Abaqus/CAE uses linear extrapolation to calculate the nodal values of element variables. To display a contour plot of Mises stress, Abaqus/CAE extrapolates the stress components from the integration points to the nodal locations within each element and calculates the Mises stress. If the differences in Mises stress values fall within the specified averaging threshold, nodal averaged Mises stresses are calculated from each surrounding element's invariant stress value. Invariant values exceeding the elastic limit can be produced by the extrapolation process.

Try plotting contours of each component of the stress tensor (variables S11, S22, S33, S12, S23, and S13). You will see that there are significant variations in these stresses across the elements at the built-in end. This causes the extrapolated nodal stresses to be higher than the values at the integration points. The Mises stress calculated from these values will, therefore, also be higher.



Note:

Surface integration elements (C3D8S, C3D8HS, and C3D10HS) do not suffer from this extrapolation problem. The integration points of these element types are located at the nodes, resulting in improved surface stress visualization.

The Mises stress at an integration point can never exceed the current yield stress of the element's material; however, the extrapolated nodal values reported in a contour plot may do so. In addition, the individual stress components may have magnitudes that exceed the value of the current yield stress; only the Mises stress is required to have a magnitude less than or equal to the value of the current yield stress.

You can use the query tools in the Visualization module to check the Mises stress at the integration points.

To query the Mises stress:

1. From the main menu bar, select **Tools->Query**; or use the  tool in the Query toolset.

   The **Query** dialog box appears.
2. In the **Visualization Module Queries** field, select **Probe values**.

   The **Probe Values** dialog box appears.
3. Make sure that **Elements** and the output position **Integration Pt** are selected.
4. Use the cursor to select elements near the constrained end of the lug.

   Abaqus/CAE reports the element ID and type by default and the value of the Mises stress at each integration point starting with the first integration point. The Mises stress values at the integration points are all lower than the values reported in the contour legend and also below the yield stress of 580 MPa. You can click mouse button 1 to store probed values.
5. Click **Cancel** when you have finished probing the results.

The fact that the extrapolated values are so different from the integration point values indicates that there is a rapid variation of stress across the elements and that the mesh is too coarse for accurate stress calculations. This extrapolation error will be less significant if the mesh is refined but will always be present to some extent. Therefore, always use nodal values of element variables with caution.

Contour plot of equivalent plastic strain

The equivalent plastic strain in a material (PEEQ) is a scalar variable that is used to represent the material's inelastic deformation. If this variable is greater than zero, the material has yielded. Those parts of the lug that have yielded can be identified in a contour plot of PEEQ by selecting **Primary** from the list of variable types on the left side of the **Field Output** toolbar and selecting **PEEQ** from the list of output variables. Any areas in the model plotted in a dark color in Abaqus/CAE still have elastic material behavior (see *Figure 3*).

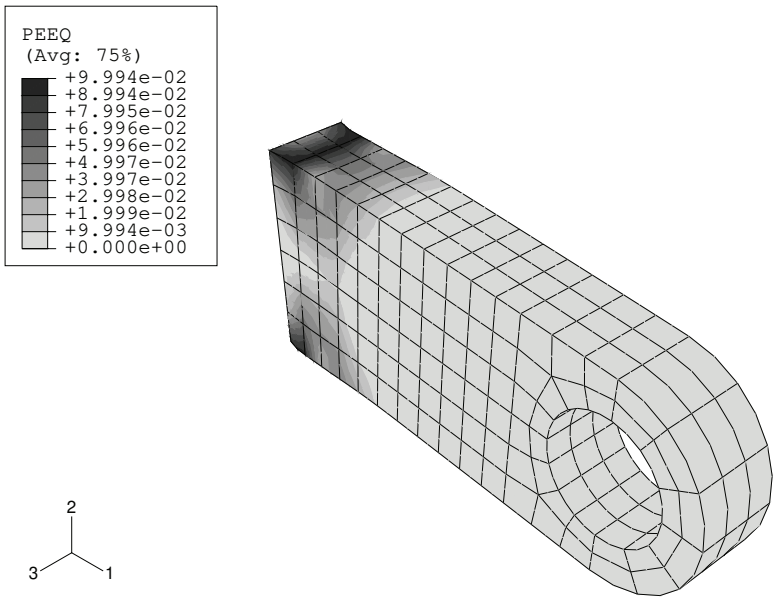


Figure 3: Contour of equivalent plastic strain (PEEQ).

It is clear from the plot that there is gross yielding in the lug where it is attached to its parent structure. The maximum plastic strain reported in the contour legend is about 10%. However, this value may contain errors from the extrapolation process. Use the query tool to check the integration point values of PEEQ in the elements with the largest plastic strains. You will find that the largest equivalent plastic strains in the model are about 0.087 at the integration points. This does not necessarily indicate a large extrapolation error since there are strain gradients present in the vicinity of the peak plastic deformation.

Creating a variable-variable (stress-strain) plot

The *X–Y* plotting capability in Abaqus/CAE was introduced earlier in this guide. In this section you will learn how to create *X–Y* plots showing the variation of one variable as a function of another. You will use the stress and strain data saved to the output database (`.odb`) file (in the form of field output rather than history output) to create a stress-strain plot for one of the integration points in an element adjacent to the constrained end of the lug.

Consider the shaded element shown in *Figure 4*.

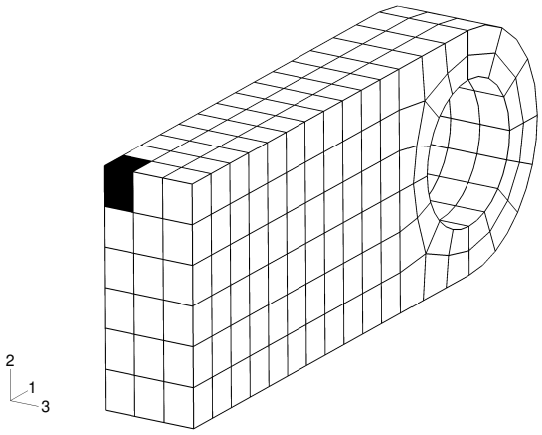


Figure 4: Element where stress and strain histories will be studied.

We will plot the stress and strain histories at an integration point in this element. The selected integration point should be as close as possible to the top surface of the lug but not adjacent to the constrained nodes. The numbering of the integration points depends on the element's nodal connectivity. Thus, you will need to identify the element's label as well as its nodal connectivity to determine which integration point to use.

To determine the integration point number:

1. In the Display Group toolbar, select the **Replace Selected** tool and click the shaded element shown in *Figure 4*.

2. Plot the undeformed shape of this element with the node labels made visible. Click the auto-fit tool to obtain a plot similar to *Figure 5*.

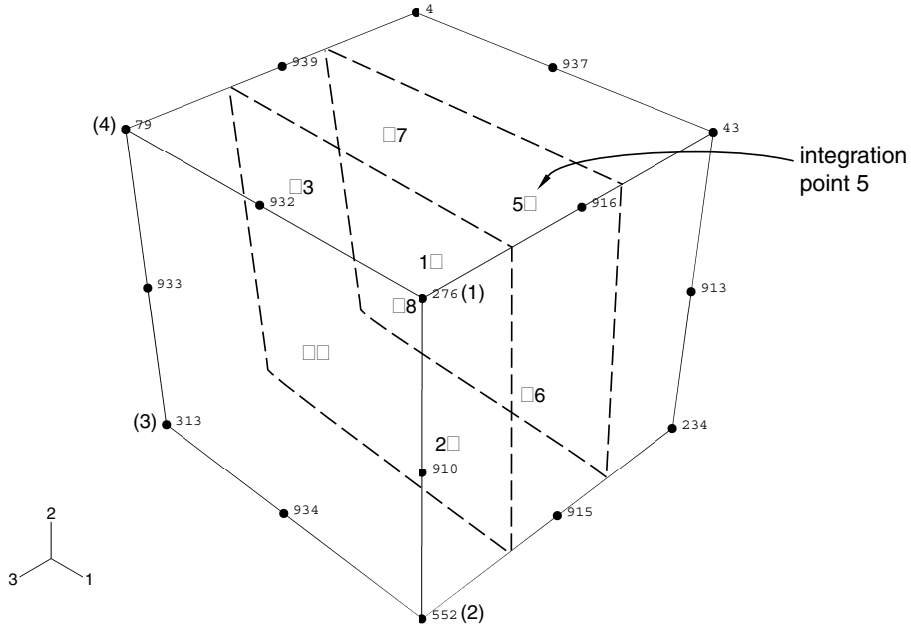


Figure 5: Location of integration point near the top surface.

3. Use the **Query** tool to obtain the nodal connectivity for this corner element (select **Element**, and click the element in the viewport). The nodal connectivity will be printed to the message area; you are interested in only the first four nodes.
4. Compare the nodal connectivity list with the undeformed model shape plot to determine which is the 1–2–3–4 face on your C3D20R element, as defined in *Three-Dimensional Solid Element Library*. For example, in *Figure 5* the 276–552–313–79 face corresponds to the 1–2–3–4 face. Thus, the integration points are numbered as shown in the figure. We are interested in the point that corresponds to integration point 5.

In the following discussion, assume that the element label is 41 and that integration point 5 of this element satisfies the requirements stated above. Your element and/or integration point numbers may differ.

To create history curves of stress and direct strain along the lug:

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.
2. In this dialog box, select **ODB field output** as the source and click **Continue**.

   The **XY Data from ODB Field Output** dialog box appears; the **Variables** tabbed page is open by default.
3. In this dialog box, expand the following lists: **S: Stress components** and **E: Strain components**.
4. From the list of available stress and strain components, select **Mises** and **E11**, respectively.

   The Mises stress, rather than the component of the true stress tensor, is used because the plasticity model defines plastic yield in terms of Mises stress. The E11 strain component is used because it is the largest component of the total strain tensor at this point; using it clearly shows the elastic, as well as the plastic, behavior of the material at this integration point.
5. Click the **Elements/Nodes** tab.
6. Accept **Pick from viewport** as the selection method, and click **Edit Selection**.
7. In the viewport, click the shaded element shown in *Figure 4*, then click **Done** in the prompt area.
8. Click **Save** to save the data followed by **Dismiss** to close the dialog box.

Sixteen curves are created (one for each variable at each integration point), and default names are given to the curves. The curves appear in the **XYData** container. Each of the curves is a history (variable versus time) plot. You must combine the plots for the integration point of interest, eliminating the time dependence, to produce the desired stress-strain plot.

To combine history curves to produce a stress-strain plot:

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.
2. Select **Operate on XY data**, and click **Continue**.

   The **Operate on XY Data** dialog box appears. Expand the **Name** field to see the full name of each curve.
3. From the **Operators** listed, select **combine(X,X)**.

   `combine( )` appears in the text field at the top of the dialog box.
4. In the **XY Data** field, select the stress and strain curves for the integration point of interest.
5. Click **Add to Expression**. The expression `combine("E:E11 ...", "S:MISES ...")` appears in the text field. In this expression `"E:E11 ..."` will determine the *X*-values and `"S:MISES ..."` will determine the *Y*-values in the combined plot.
6. Save the combined data object by clicking **Save As** at the bottom of the dialog box.

The **Save XY Data As** dialog box appears. In the **Name** text field, type `SVE11`; and click **OK** to close the dialog box.

7. To view the combined stress-strain plot, click **Plot Expression** at the bottom of the dialog box.
8. Click **Cancel** to close the dialog box.
9. Click  in the prompt area to cancel the current procedure.

This *X*–*Y* plot would be clearer if the limits on the *X*- and *Y*-axes were changed.

To customize the stress-strain curve:

1. Double-click either axis to open the **Axis Options** dialog box.
2. Set the maximum value of the *X*-axis (E11 strain) to `0.09`, the maximum value of the *Y*-axis (MISES stress) to `500` MPa, and the minimum value to `0.0` MPa.
3. Switch to the **Title** tabbed page, and customize the *X*- and *Y*-axis labels as shown in *Figure 6*.

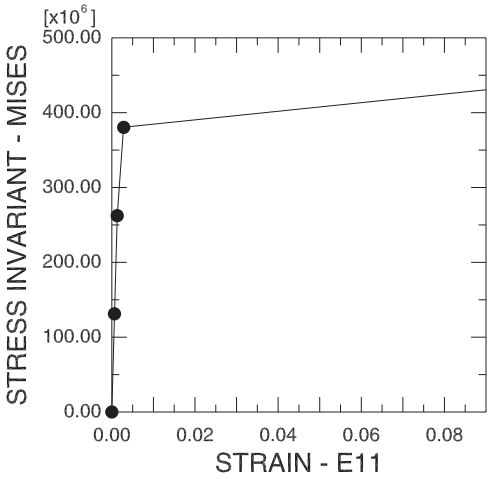


Figure 6: Mises stress vs. direct strain (E11) along the lug in the corner element.

4. Click **Dismiss** to close the **Axis Options** dialog box.
5. It will also be helpful to display a symbol at each data point of the curve. Open the **Curve Options** dialog box.
6. From the **Curves** field, select the stress-strain curve (`SVE11`).

   The `SVE11` data object is highlighted.
7. Toggle on **Show symbol**. Accept the defaults, and click **Dismiss** at the bottom of the dialog box.

   The stress-strain plot appears with a symbol at each data point of the curve.

You should now have a plot similar to the one shown in *Figure 6*. The stress-strain curve shows that the material behavior was linear elastic for this integration point during the first two increments of the simulation. In this plot it appears that the material remains linear during the third increment of the analysis; however, it does yield during this increment. This illusion is created by the extent of strain shown in the plot. If you limit the maximum strain displayed to 0.01 and set the minimum value to 0.0, the nonlinear material behavior in the third increment can be seen more clearly (see *Figure 7*).

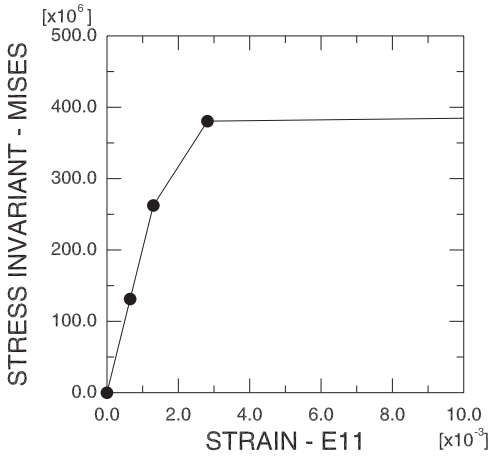


Figure 7: Mises stress vs. direct strain (E11) along the lug in the corner element. Maximum strain 0.01.

This stress-strain curve contains another apparent error. It appears that the material yields at 250 MPa, which is well below the initial yield stress. However, this error is caused by the fact that Abaqus/CAE connects the data points on the curve with straight lines. If you limit the increment size, the additional points on the graph will provide a better display of the material response and show yield occurring at exactly 380 MPa.

The results from this second simulation indicate that the lug will withstand this 60 kN load if the steel hardens after it yields. Taken together, the results of the two simulations demonstrate that it is very important to determine the actual post-yield hardening behavior of the steel. If the steel has very little hardening, the lug may collapse under the 60 kN load. Whereas if it has moderate hardening, the lug will probably withstand the load although there will be extensive plastic yielding in the lug (see *Figure 3*). However, even with plastic hardening, the factor of safety for this loading will probably be very small.

Example: blast loading on a stiffened plate

In this example you will solve a problem involving plasticity using explicit dynamics.

In this section:

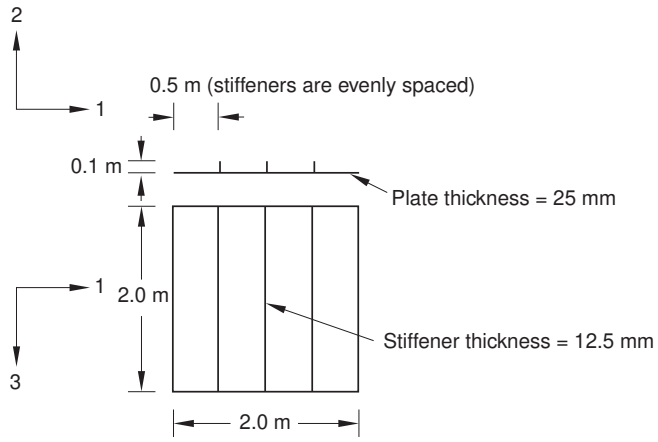
- *Blast loading on a stiffened plate analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*
- *Reviewing the analysis*

Blast loading on a stiffened plate analysis

In this example you will assess the response of a stiffened square plate subjected to a blast loading in Abaqus/Explicit.

The previous example illustrated some of the convergence difficulties that may be encountered when solving problems involving a nonlinear material response using implicit methods. We will now focus on solving a problem involving plasticity using explicit dynamics. As will become evident shortly, convergence difficulties are not an issue in this case since iteration is not required for explicit methods.

The plate is firmly clamped on all four sides and has three equally spaced stiffeners welded to it. The plate is constructed of 25 mm thick steel and is 2 m square. The stiffeners are made from 12.5 mm thick plate and have a depth of 100 mm. *Figure 1* shows the plate geometry and material properties in more detail. Since the plate thickness is significantly smaller than any other global dimensions, shell elements can be used to model the plate.



Material properties

General properties:

$\rho = 7800\ kg/m^3$

Elastic properties:

$E = 210 \times 10^9$ Pa

$\nu = 0.3$

Plastic properties:

True Stress (Pa)	True Plastic Strain
300×10^6	0.000
350×10^6	0.025
375×10^6	0.100
394×10^6	0.200
400×10^6	0.350

Figure 1: Problem description for blast load on a stiffened plate.

The purpose of this example is to determine the response of the plate and to see how it changes as the sophistication of the material model increases. Initially, we analyze the behavior with the standard elastic-plastic material model. Subsequently, we study the effects of including material damping and rate-dependent material properties.

Preprocessing—creating the model with Abaqus/CAE

Use Abaqus/CAE to create a three-dimensional model of the stiffened plate.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Blast loading on a stiffened plate*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Blast loading on a stiffened plate**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Defining the model geometry

Create a three-dimensional, deformable part with an extruded shell base feature to represent the plate. Use an approximate part size of `5.0`, and name the part `Plate`. A suggested approach for creating the part geometry shown in *Figure 1* is summarized in the following procedure:

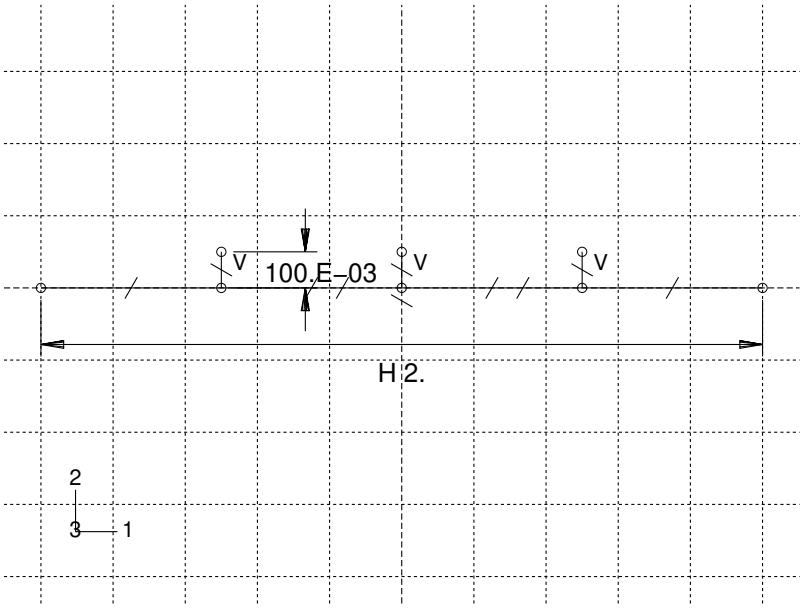


Figure 1: Sketch of the stiffened plate (with grid spacing doubled).

To create the stiffened plate geometry:

1. To define the plate geometry, use the **Create Lines: Connected** tool to sketch an arbitrary horizontal line.
2. To define the stiffener geometry, add three vertical lines extending up from the plate. The horizontal position of these lines is arbitrary at this stage, but their endpoints must snap to the horizontal line.
3. Constrain the three vertical lines so they are of equal length, and dimension one of them so that it is 0.1 m long.
4. Split the plate at the points where it intersects the stiffeners.
5. Apply horizontal constraints to each of the horizontal segments of the line.

6. Apply equal length constraints to the four horizontal segments of the line.
7. Dimension the horizontal distance between the plate endpoints, and set the value to 2.0 m.

   The final part sketch is shown in *Figure 1*.
8. Extrude the sketch to a depth of 2.0 m to create the plate.

Defining the material properties

Define the material and section properties for the plate and the stiffeners.

Create a material named `Steel` with a mass density of `7800` kg/m^3, a Young's modulus of `210.0E9` Pa, and a Poisson's ratio of `0.3`. At this stage we do not know whether there will be any plastic deformation, but we know the value of the yield stress and the details of the post-yield behavior for this steel. We will include this information in the material definition. The initial yield stress is 300 MPa, and the yield stress increases to 400 MPa at a plastic strain of 35%. To define the plastic material properties, enter the yield stress and plastic strain data shown in *Figure 1*. The plasticity stress-strain curve is shown in *Figure 2*.

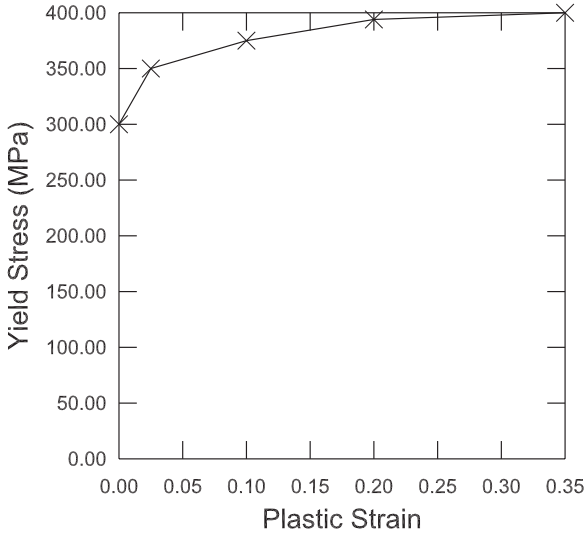


Figure 2: Yield stress versus plastic strain.

During the analysis Abaqus calculates values of yield stress from the current values of plastic strain. As discussed earlier, the process of lookup and interpolation is most efficient when the stress-strain data are at equally spaced values of plastic strain. To avoid having the user input regular data, Abaqus/Explicit automatically regularizes the data. In this case the data are regularized by Abaqus/Explicit by expanding the data to 15 equally spaced points with increments of 0.025.

To illustrate the error message that is produced when Abaqus/Explicit cannot regularize the material data, you could set the regularization tolerance to 0.001 (in the **Edit Material** dialog box, select **General->Regularization**) and include one additional data pair, as shown in *Table 1*. You can add a row by clicking mouse button 3 in the table and selecting **Insert Row** from the menu that appears.

Table 1: Modified plasticity data.

Yield Stress (Pa)	Plastic Strain
300.0E6	0.000
349.0E6	0.001
350.0E6	0.025

Yield Stress (Pa)	Plastic Strain
375.0E6	0.100
394.0E6	0.200
400.0E6	0.350

The combination of the low tolerance value and the small interval in the user-defined data would lead to difficulty in regularizing this material definition. The following error message would be written to the status (`.sta`) file and displayed in the **Job Monitor** dialog box in the Job module:

```
***ERROR: Failed to regularize material data for material
STEEL. Please check your input data to see if they meet
both criteria as explained in "MATERIAL DATA DEFINITION"
section of the Abaqus Materials Guide. In general,
regularization is more difficult if the smallest interval
defined by the user is small compared to the range of the
independent variable.
```

Before continuing, set the regularization tolerance back to the default value (0.03) and remove the additional pair of data points.

Creating and assigning section properties

Create two homogeneous shell section properties, each referring to the steel material definition but specifying different shell thicknesses. Name the first shell section property `PlateSection`, select `Steel` as the material, and specify `0.025` m as the value for the **Shell thickness**. Name the second shell section property `StiffSection`, select `Steel` as the material, and specify `0.0125` m as the value for the **Shell thickness**.

Assign the `StiffSection` definition to the stiffeners (use **[Shift] + Click** to select multiple regions in the viewport).

Before assigning the `PlateSection` definition to the plate, consider the following. If the plate and the stiffeners are joined directly at their midsurfaces (this is the default behavior), an area of material overlap will occur, as shown in *Figure 3*.

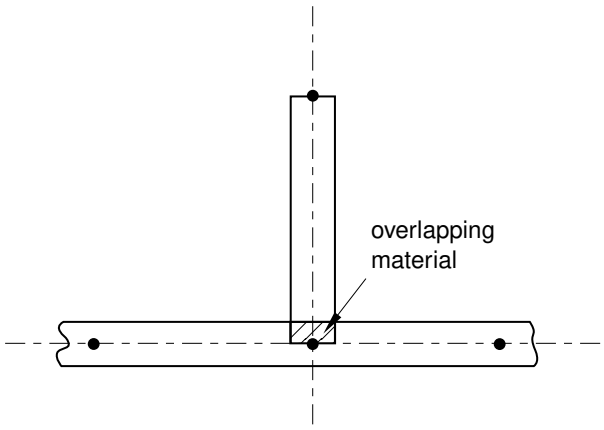


Figure 3: Overlapping material.

Although the thicknesses of the plate and stiffener are small in comparison to the overall dimensions of the structure (so that this overlapping material and the extra stiffness it creates would have little effect on the analysis results), a more precise model can be created by offsetting the plate reference surface from its midsurface. This technique allows the stiffeners to butt up against the plate without overlapping any material with the plate, as shown in *Figure 4*.

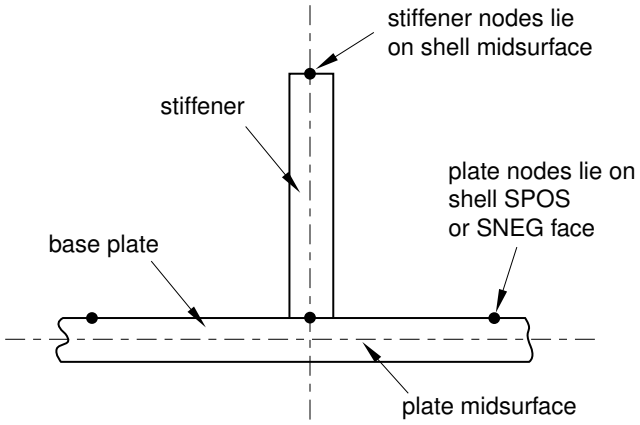


Figure 4: Stiffener joint in which the plate's reference surface is offset from its midsurface.

To determine whether to offset the plate reference surface to its positive (SPOS) or negative (SNEG) side, query the shell normals (**Tools->Query**) and note the color of the side of the plate facing the stiffeners (brown is the positive side; purple is the negative side). If necessary, flip the plate normals (**Assign->Element Normal**) so that its segments have consistent normals. Then assign the `PlateSection` definition to the regions of the plate. In the **Edit Section Assignment** dialog box, set the shell offset to **Top surface** if the brown (positive) side of the plate faces the stiffeners and **Bottom surface** if the purple (negative) side faces the stiffeners.

To verify the offset, select **View->Part Display Options**. In the **Part Display Options** dialog box that appears, toggle on **Render shell thickness**. If necessary, modify the offset to remove any overlap.

The model can be color-coded according to section assignment to verify that properties were assigned correctly (select **Sections** from the **Color Code** toolbar).

Creating an assembly

Create a dependent instance of the plate. Use the default rectangular coordinate system, with the plate lying in the 1–3 plane.

At this point, it is convenient to create the geometry sets that will be used to specify boundary conditions and output requests. Create an assembly-level set named `Edge` for the plate edges and another named `Center` at the center of the intersection of the plate and the middle stiffener, as shown in *Figure 5*. To create the set `Center`, you need to first partition the edge of the original part in half using the **Partition Edge: Enter Parameter** tool (the partition must be created in either the Part or Property module since this is a dependent part instance).

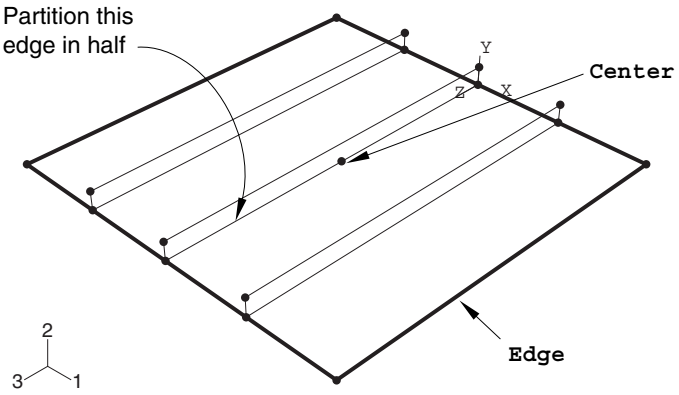


Figure 5: Geometry sets.

Defining steps and output requests

Create a single dynamic, explicit step. Name the step `Blast`, and specify the following step description: `Apply blast loading`. Enter a value of `50E-3` s for the time period of the step.

In general, you should try to limit the number of frames written during the analysis to keep the size of the output database file reasonable. In this analysis saving information every 2 ms should provide sufficient detail to study the response of the structure. Edit the default output request `F-Output-1`, and set the number of intervals during the step at which preselected field data are saved to 25. This ensures that the selected data are written every 2 ms since the total time for the step is 50 ms.

A more detailed set of output for selected regions of the model can be saved as history output. Create a history output request named `Center-U2` for the step `Blast`. Select `Center` as the output domain, and select **U2** as the translation output variable. Enter `500` as the number of intervals at which the output will be saved during the analysis.

Prescribing boundary conditions and loads

Next, define the boundary conditions used in this analysis. In the step `Blast`, create a **Symmetry/Antisymmetry/Encastre** mechanical boundary condition named `Fix edges`. Apply the boundary condition to the edges of the plate using the geometry set `Edge`, and specify **ENCASTRE (U1 = U2 = U3 = UR1 = UR2 = UR3 = 0)** to fully constrain the set.

The plate will be subjected to a load that varies with time: the pressure increases rapidly from zero at the start of the analysis to its maximum of 7.0×10^5 Pa in 1 ms, at which point it remains constant for 9 ms before dropping back to zero in another 10 ms. It then remains at zero for the remainder of the analysis. See *Figure 6* for details.

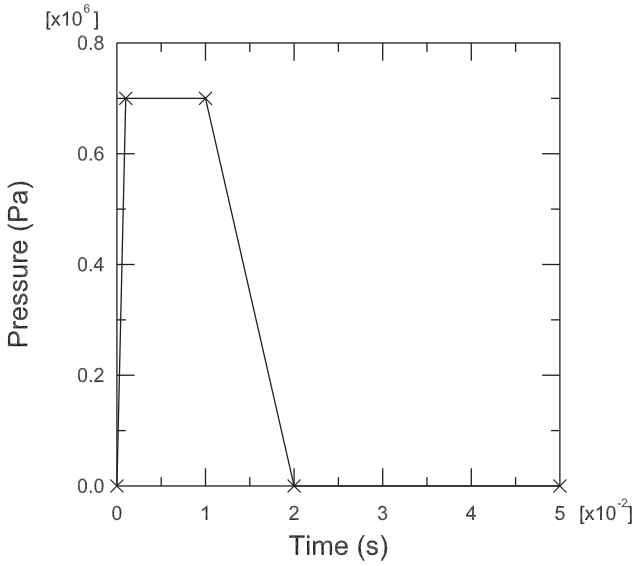


Figure 6: Pressure load as a function of time.

Define a tabular amplitude curve named `Blast`. Enter the amplitude data given in *Table 2*, and specify a smoothing parameter of 0.0.

Table 2: Blast load amplitude.

Time	Amplitude
0.0	0.0

Time	Amplitude
1.0E−3	7.0E5
10.0E−3	7.0E5
20.0E−3	0.0
50.0E−3	0.0

Next, define the pressure loading. Since the magnitude of the load will be defined by the amplitude definition, you need to apply only a unit pressure to the plate. Apply the pressure so that it pushes against the top of the plate (where the stiffeners are on the bottom of the plate). Such a pressure load will place the outer fibers of the stiffeners in tension.

To define the pressure loading:

1. In the Model Tree, double-click the **Loads** container. In the **Create Load** dialog box that appears, name the load `Pressure load` and select `Blast` as the step in which it will be applied. Select **Mechanical** as the load category and **Pressure** as the load type. Click **Continue**.
2. Select all the surfaces associated with the plate. When the appropriate surfaces are selected, click **Done**.

   Abaqus/CAE uses two different colors to indicate the two sides of the shell surface. To complete the load definition, the colors must be consistent on each side of the plate.
3. If necessary, select **Flip a surface** in the prompt area to flip the colors for a region of the plate. Repeat this procedure until all of the faces on the top of the plate are the same color.
4. In the prompt area, select the color representing the side of the plate without the stiffeners.
5. In the **Edit Load** dialog box that appears, specify a uniform pressure of `1.0` Pa, and select the amplitude definition `Blast`. Click **OK** to complete the load definition.

The plate load and boundary conditions are shown in *Figure 7*.

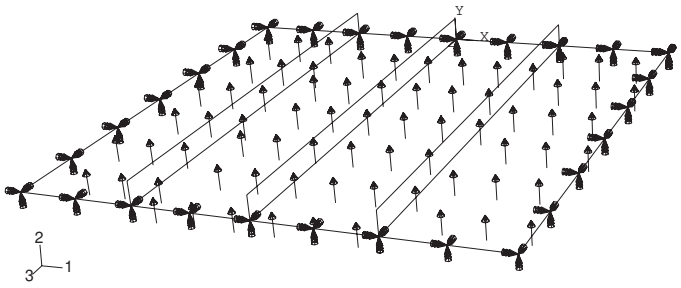


Figure 7: Pressure load and boundary conditions.

Creating the mesh and defining a job

Seed the part with a global element size of `0.1`. In addition, select **Seed->Edges** and specify that two elements be created along the height of each stiffener (in the **Local Seeds** dialog box, select **By number** as the method and set the number of elements to `2`; toggle on the option to create a set containing the selected edges). Mesh the plate and stiffeners using quadrilateral shell elements (S4R) from the **Explicit** element library. The resulting mesh is shown in *Figure 8*. This relatively coarse mesh provides moderate accuracy while keeping the solution time to a minimum.

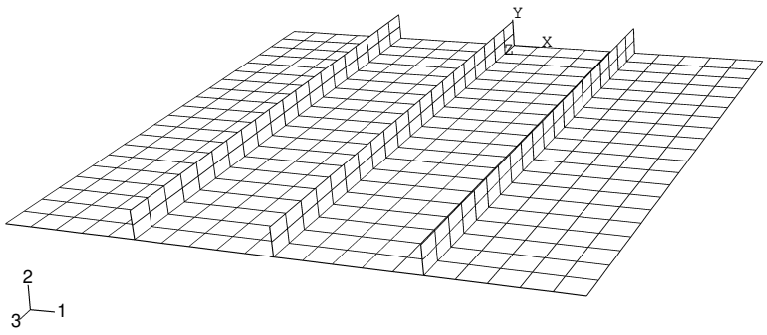


Figure 8: Meshed plate.

Create a job named `BlastLoad`. Specify the following job description: `Blast load on a flat plate with stiffeners: S4R elements (20x20 mesh) Normal stiffeners (20x2).`

Save your model in a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate the cause of any warning messages.

Postprocessing

When the job completes, enter the Visualization module, and open the `.odb` file created by this job (`BlastLoad.odb`). By default, Abaqus plots the undeformed model shape with the shaded render style.

Changing the view

The default view is isometric, which does not provide a particularly clear view of the plate. To improve the viewpoint, rotate the view using the options in the **View** menu or the tools in the View Manipulation toolbar. Specify the view and select the viewpoint method for rotating the view. Enter the *x*-, *y*-, and *z*-coordinates of the viewpoint vector as `1,0.5,1` and the coordinates of the up vector as `0,1,0`.

Verifying shell section assignment

You can also visualize the section assignment and the shell thickness while postprocessing the results. For example, regions with common section assignments can be color coded to verify that the properties were assigned correctly (select **Sections** from the **Color Code** toolbar to color the mesh according to section assignment). To render the shell thickness, select **View->ODB Display Options** from the main menu bar. In the **ODB Display Options** dialog box, toggle on **Render shell thickness** and click **Apply**. If the model looks correct, as shown in *Figure 1*, toggle off this option and click **OK** before proceeding with the rest of the postprocessing instructions. Otherwise, correct the section assignment and rerun the job.

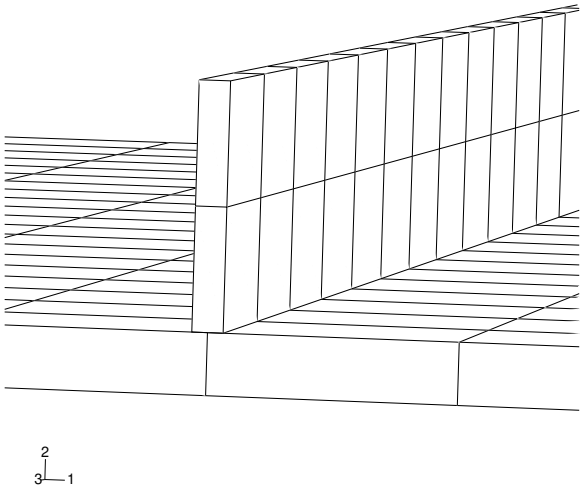


Figure 1: Plate with shell thickness displayed.

Animation of results

As noted in earlier examples, animating your results will provide a general understanding of the dynamic response of the plate under the blast loading. First, plot the deformed model shape. Then, create a time-history animation of the deformed shape. Use the **Animation Options** dialog box to change the mode to **Play once**.

You will see from the animation that as the blast loading is applied, the plate begins to deflect. Over the duration of the load the plate begins to vibrate and continues to vibrate after the blast load has dropped to zero. The maximum displacement occurs at approximately 8 ms, and a displaced plot of that state is shown in *Figure 2*.

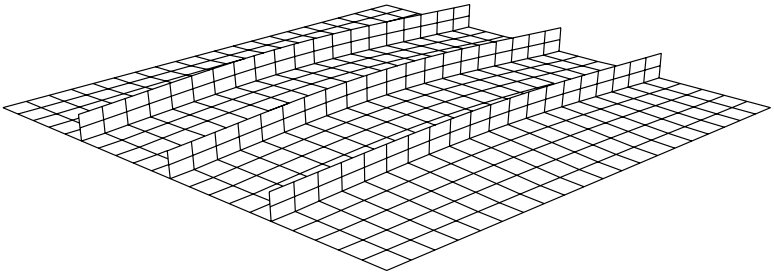


Figure 2: Displaced shape at 8 ms.

The animation images can be saved to a file for playback at a later time.

To save the animation:

1. From the main menu bar, select **Animate->Save As**.

   The **Save Image Animation** dialog box appears.
2. In the **Settings** field, enter the file name `blast_base`.

   The format of the animation can be specified as MP4, AVI, QuickTime, GIF, VRML, or Compressed VRML.
3. Choose the **QuickTime** format, and click **OK**.

   The animation is saved as `blast_base.mov` in your current directory. Once saved, your animation can be played external to Abaqus/CAE using industry-standard animation software.

History output

Since it is not easy to see the deformation of the plate from the deformed plot, it is desirable to view the deflection response of the central node in the form of a graph. The displacement of the node in the center of the plate is of particular interest since the largest deflection occurs at this node.

Display the displacement history of the central node, as shown in *Figure 3* (with displacements in millimeters).

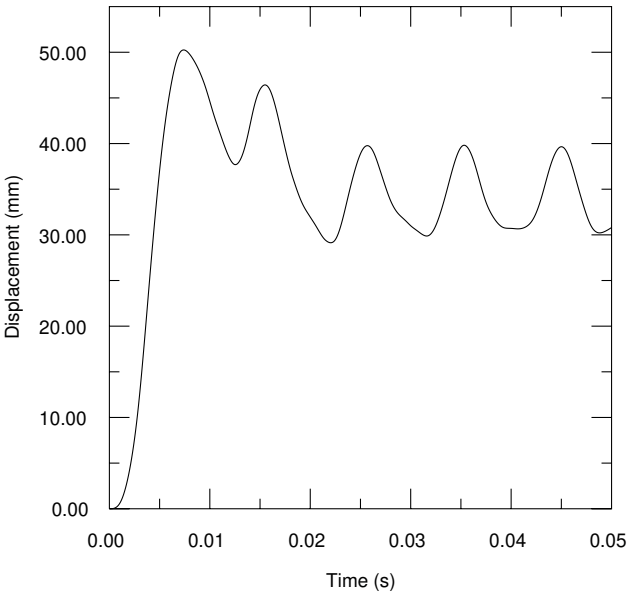


Figure 3: Central node displacement as a function of time.

To generate a history plot of the central node displacement:

1. In the Results Tree, double-click the history output data named `Spatial displacement: U2` at the node in the center of the plate (set `Center`).
2. Save the current *X*–*Y* data: in the Results Tree, click mouse button 3 on the data name and select **Save As** from the menu that appears. Name the data `DISP`.

   The units of the displacements in this plot are meters. Modify the data to create a plot of displacement (in millimeters) versus time by creating a new data object.
3. In the Results Tree, expand the **XYData** container.

   The `DISP` data are listed underneath.
4. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
5. In the **Operate on XY Data** dialog box, multiply `DISP` by 1000 to create the plot with the displacement values in millimeters instead of meters. The expression at the top of the dialog box should appear as:

   ```
   "DISP" * 1000
   ```
6. Click **Plot Expression** to see the modified *X*–*Y* data. Save the data as `U2_BASE`.
7. Close the **Operate on XY Data** dialog box.
8. Click the **Axis Options** tool in the toolbox. In the **Axis Options** dialog box, change the *X*-axis title to `Time (s)` and the *Y*-axis title to `Displacement (mm)`. Click **OK** to close the dialog box. The resulting plot is shown in *Figure 3*.

   The plot shows that the displacement reaches a maximum of 50.2 mm at 7.7 ms and then oscillates after the blast load is removed.

The other quantities saved as history output in the output database are the total energies of the model. The energy histories can help identify possible shortcomings in the model as well as highlight significant physical effects. Display the histories of five different energy output variables—ALLAE, ALLIE, ALLKE, ALLPD, and ALLSE.

To generate history plots of the model energies:

1. Save the history results for the ALLAE, ALLIE, ALLKE, ALLPD, and ALLSE output variables as *X*–*Y* data. A default name is given to each curve; rename each according to its output variable name: `ALLAE`, `ALLKE`, etc.
2. In the Results Tree, expand the **XYData** container.

   The `ALLAE`, `ALLIE`, `ALLKE`, `ALLPD`, and `ALLSE`*X*–*Y* data objects are listed underneath.
3. Select `ALLAE`, `ALLIE`, `ALLKE`, `ALLPD`, and `ALLSE` using **[Ctrl] + Click**; click mouse button 3, and select **Plot** from the menu that appears to plot the energy curves.
4. To more clearly distinguish between the different curves in the plot, open the **Curve Options** dialog box and change their line styles.
   - For the curve `ALLSE`, select a dashed line style.
   - For the curve `ALLPD`, select a dotted line style.
   - For the curve `ALLAE`, select a chain dashed line style.
   - For the curve `ALLIE`, select the second thinnest line type.

5. To change the position of the legend, open the **Chart Legend Options** dialog box and switch to the **Area** tabbed page.
6. In the **Position** region of this page, toggle on **Inset** and click **Dismiss**. Drag the legend in the viewport so that it fits within the grid, as shown in *Figure 4*.

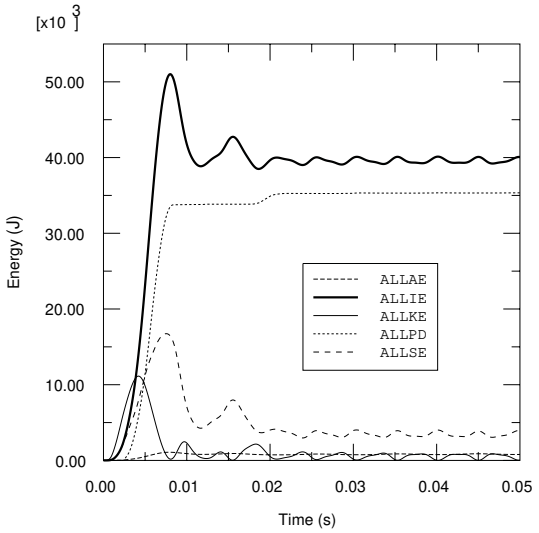


Figure 4: Energy quantities as a function of time.

We can see that once the load has been removed and the plate vibrates freely, the kinetic energy increases as the strain energy decreases. When the plate is at its maximum deflection and, therefore, has its maximum strain energy, it is almost entirely at rest, causing the kinetic energy to be at a minimum.

The plastic strain energy rises to a plateau and then rises again. From the plot of kinetic energy we can see that the second rise in plastic strain energy occurs when the plate has rebounded from its maximum displacement and is moving back in the opposite direction. We are, therefore, seeing plastic deformation on the rebound after the blast pulse.

Even though there is no indication that hourglassing is a problem in this analysis, study the artificial strain energy to make sure. As discussed in *Using Continuum Elements*, artificial strain energy or "hourglass stiffness" is the energy used to control hourglass deformation, and the output variable ALLAE is the accumulated artificial strain energy. This discussion on hourglass control applies equally to shell elements. Since energy is dissipated as plastic deformation as the plate deforms, the total internal energy is much greater than the elastic strain energy alone. Therefore, it is most meaningful in this analysis to compare the artificial strain energy to an energy quantity that includes the dissipated energy as well as the elastic strain energy. Such a variable is the total internal energy, ALLIE, which is a summation of all internal energy quantities. The artificial strain energy is approximately 2% of the total internal energy, indicating that hourglassing is not a problem.

One thing we can notice from the deformed shape is that the central stiffener is subject to almost pure in-plane bending. Using only two first-order, reduced-integration elements through the depth of the stiffener is not sufficient to model in-plane bending behavior. While the solution from this coarse mesh appears to be adequate since there is little hourglassing, for completeness we will study how the solution changes when we refine the mesh of the stiffener. Use caution when you refine the mesh, since mesh refinement will increase the solution time by increasing the number of elements and decreasing the element size.

Edit the mesh, and respecify the mesh density. Using the previously saved edge set, specify four elements through the height of each stiffener, and remesh the part instance. Create a new job named `BlastLoadRefined`. Submit this job for analysis, and investigate the results when the job has completed running.

This increase in the number of elements increases the solution time by approximately 20%. In addition, the stable time increment decreases by approximately a factor of two as a result of the reduction of the smallest element

dimension in the stiffeners. Since the total increase in solution time is a combination of the two effects, the solution time of the refined mesh increases by approximately a factor of 1.2×2, or 2.4, over that of the original mesh.

Figure 5 shows the histories of artificial energy for both the original mesh and the mesh with the refined stiffeners. The artificial energy is slightly lower in the refined mesh. As a result, we would not expect the results to change significantly from the original to the refined mesh.

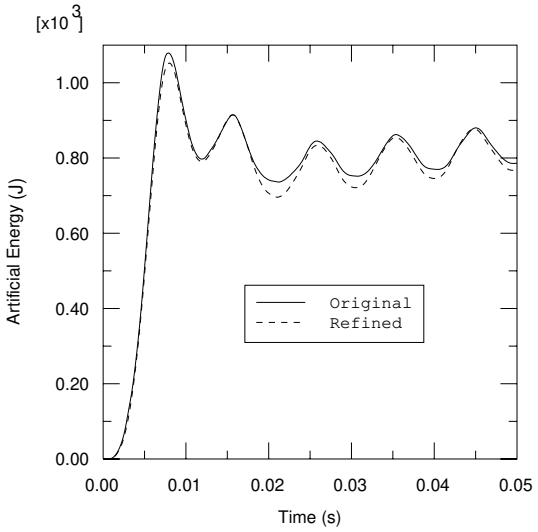


Figure 5: Artificial energy in the original and refined meshes.

Figure 6 shows that the displacement of the plate's central node is almost identical in both cases, indicating that the original mesh is capturing the overall response adequately. One advantage of the refined mesh, however, is that it better captures the variation of stress and plastic strain through the stiffeners.

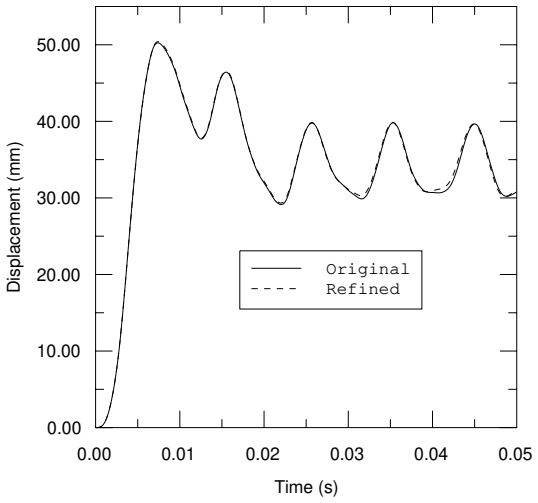


Figure 6: Central node displacement history for the original and refined meshes.

Contour plots

In this section you will use the contour plotting capability of the Visualization module to display the von Mises stress and equivalent plastic strain distributions in the plate. Use the model with the refined stiffener mesh to create the plots; from the main menu bar, select **File->Open** and choose the file `BlastLoadRefined.odb`.

To generate contour plots of the von Mises stress and equivalent plastic strain:

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Primary**.
2. From the list of output variables in the center of the toolbar, select **S**. The stress invariants and components are available in the next list to the right. Select the **Mises** stress invariant.
3. From the main menu bar, select **Result->Section Points**.
4. In the **Section Points** dialog box that appears, select **Top and bottom** as the active locations and click **OK**.
5. Select **Plot->Contours->On Deformed Shape**, or use the  tool from the toolbox.

   Abaqus plots the contours of the von Mises stress on both the top and bottom faces of each shell element. To see this more clearly, rotate the model in the viewport.

   The view that you set earlier for the animation exercise should be changed so that the stress distribution is clearer.
6. Change the view back to the default isometric view using the  tool in the Views toolbar.

    **Tip:** If the Views toolbar is not visible, select **View->Toolbars->Views** from the main menu bar.

   Figure 7 shows a contour plot of the von Mises stress at the end of the analysis.

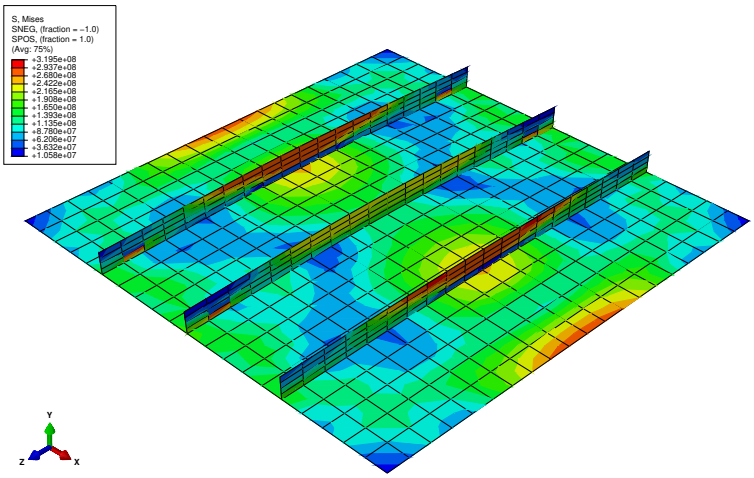


Figure 7: Contour plot of von Mises stress at 50 ms.

7. Similarly, contour the equivalent plastic strain. Select **Primary** from the list of variable types on the left side of the **Field Output** toolbar and select **PEEQ** from the list of output variables next to it.

   Figure 8 shows a contour plot of the equivalent plastic strain at the end of the analysis.

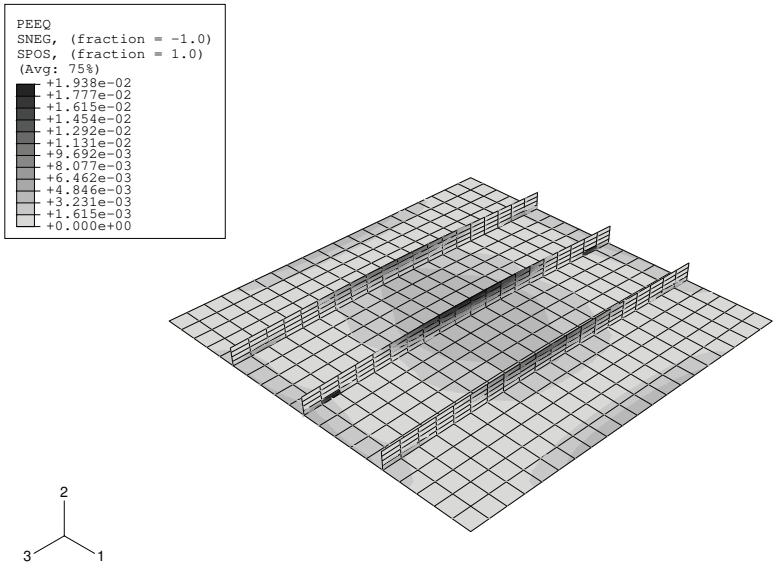


Figure 8: Contour plot of equivalent plastic strain at 50 ms.

Reviewing the analysis

The objective of this analysis is to study the deformation of the plate and the stress in various parts of the structure when it is subjected to a blast load. To judge the accuracy of the analysis, you will need to consider the assumptions and approximations made and identify some of the limitations of the model.

Damping

Undamped structures continue to vibrate with constant amplitude. Over the 50 ms of this simulation, the frequency of the oscillation can be seen to be approximately 100 Hz. A constant amplitude vibration is not the response that would be expected in practice since the vibrations in this type of structure would tend to die out over time and effectively disappear after 5–10 oscillations. The energy loss typically occurs by a variety of mechanisms including frictional effects at the supports and damping by the air.

Consequently, we need to consider the presence of damping in the analysis to model this energy loss. The energy dissipated by viscous effects, ALLVD, is nonzero in the analysis, indicating that there is already some damping present. By default, a *bulk viscosity* damping (discussed in *Nonlinear Explicit Dynamics*) is always present and is introduced to improve the modeling of high-speed events.

In this shell model only linear damping is present. With the default value the oscillations would eventually die away, but it would take a long time because the bulk viscosity damping is very small. Material damping should be used to introduce a more realistic structural response. Thus, modify the material definition.

To add damping to a material:

1. In the Model Tree, double-click **Steel** underneath the **Materials** container.
2. In the **Edit Material** dialog box, select **Mechanical->Damping** and specify `50` as the value for the mass proportional damping factor, **Alpha**. **Beta** is the parameter that controls stiffness proportional damping; at this stage, leave it set to zero.
3. Click **OK**.

The duration of the oscillation of the plate is approximately 30 ms, so we need to increase the analysis period to allow enough time for the vibration to be damped out. Edit the step definition, and increase the time period of step `Blast` to `150E-3` s.

The results of the damped analysis clearly show the effect of mass proportional damping. *Figure 1* shows the displacement history of the central node for both the damped and undamped analyses. (We have extended the analysis time to 150 ms for the undamped model to compare the data more effectively.) The peak response is also reduced due to damping. By the end of the damped analysis the oscillation has decayed to a nearly static condition.

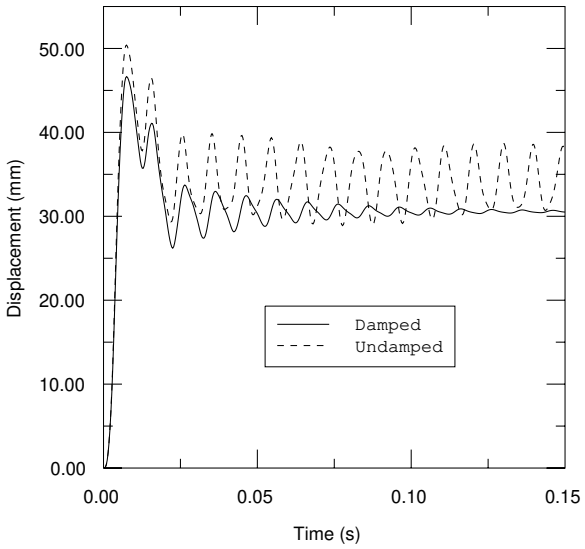


Figure 1: Damped and undamped displacement histories.

Rate dependence

Some materials, such as mild steel, show an increase in the yield stress with increasing strain rate. In this example the loading rate is high, so strain-rate dependence is likely to be important.

Add rate dependence to your material definition.

To add rate dependence to your metal plasticity material model:

1. In the Model Tree, double-click **Steel** underneath the **Materials** container.
2. In the **Edit Material** dialog box, select **Plastic** from the list of material behaviors.
3. Select **Suboptions->Rate Dependent**.
4. In the **Suboption Editor** that appears, enter a value of `40` for the **Multiplier** and a value of `5` for the **Exponent**, and click **OK**.

With this definition of rate-dependent behavior, the ratio of the dynamic yield stress to the static yield stress (R) is given for an equivalent plastic strain rate ($\dot{\bar{\varepsilon}}^{pl}$), according to the equation $\dot{\bar{\varepsilon}}^{pl} = D(R-1)^n$, where D and n are material constants (40 and 5 in this case).

Change the time period of step `Blast` back to the original value of 50 ms. Create a job named `BlastLoadRateDep`, and submit the job for analysis. When the analysis is complete, open the output database file `BlastLoadRateDep.odb`, and postprocess the results.

When rate dependence is included, the yield stress effectively increases as the strain rate increases. Therefore, because the elastic modulus is higher than the plastic modulus, we expect a stiffer response in the analysis with rate dependence. Both the displacement history of the central portion of the plate shown in *Figure 2* and the history of plastic strain energy shown in *Figure 3* confirm that the response is indeed stiffer when rate dependence is included. The results are, of course, sensitive to the material data. In this case the values of D and n are typical of a mild steel, but more precise data would be required for detailed design analyses.

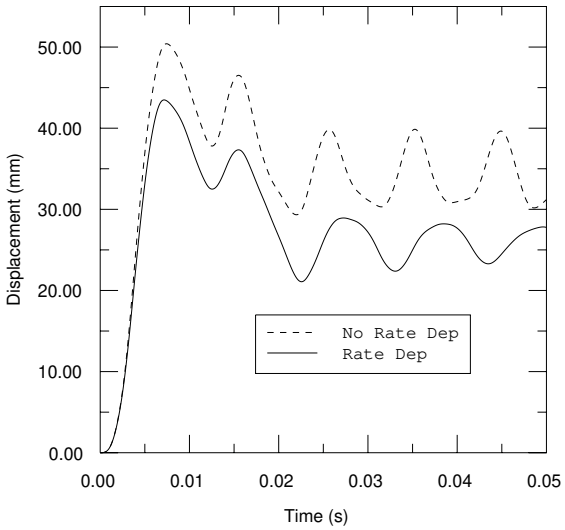


Figure 2: Displacement of the central node with and without rate dependence.

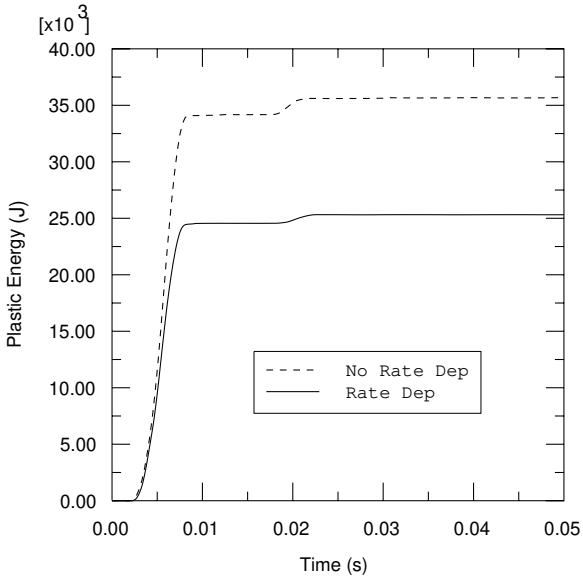


Figure 3: Plastic strain energy with and without rate dependence.

Hyperelasticity

We now turn our attention to another class of material nonlinearity, namely, the nonlinear elastic response exhibited by rubber materials.

In this section:

- *Introduction*
- *Compressibility*
- *Strain energy potential*
- *Defining hyperelastic behavior using test data*

Introduction

The stress-strain behavior of typical rubber materials, shown in *Figure 1*, is elastic but highly nonlinear.

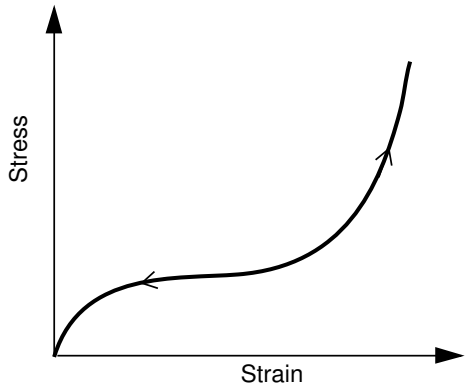


Figure 1: Typical stress-strain curve for rubber.

This type of material behavior is called *hyperelasticity*. The deformation of hyperelastic materials, such as rubber, remains elastic up to large strain values (often well over 100%).

Abaqus makes the following assumptions when modeling a hyperelastic material:

- The material behavior is elastic.
- The material behavior is initially isotropic.
- The simulation will include nonlinear geometric effects.

In addition, Abaqus/Standard assumes the hyperelastic material is incompressible by default. Abaqus/Explicit assumes the material is nearly incompressible (Poisson's ratio is 0.475 by default).

Elastomeric foams are another class of highly nonlinear, elastic materials. They differ from rubber materials in that they have very compressible behavior when subjected to compressive loads. They are modeled with a separate material model in Abaqus and are not discussed in detail in this guide.

Compressibility

Most solid rubber materials have very little compressibility compared to their shear flexibility. This behavior is not a problem with plane stress, shell, or membrane elements. However, it can be a problem when using other elements, such as plane strain, axisymmetric, and three-dimensional solid elements. For example, in applications where the material is not highly confined, it would be quite satisfactory to assume that the material is fully incompressible: the volume of the material cannot change except for thermal expansion. In cases where the material is highly confined (such as an O-ring used as a seal), the compressibility must be modeled correctly to obtain accurate results.

Abaqus/Standard has a special family of “hybrid” elements that must be used to model the fully incompressible behavior seen in hyperelastic materials. These “hybrid” elements are identified by the letter `H' in their name; for example, the hybrid form of the 8-node brick, C3D8, is called C3D8H.

Except for plane stress and uniaxial cases, it is not possible to assume that the material is fully incompressible in Abaqus/Explicit because the program has no mechanism for imposing such a constraint at each material calculation point. An incompressible material also has an infinite wave speed, resulting in a time increment of zero. Therefore, we must provide some compressibility. The difficulty is that, in many cases, the actual material behavior provides too little compressibility for the algorithms to work efficiently. Thus, except for plane stress and uniaxial cases, the user must provide enough compressibility for the code to work, knowing that this makes the bulk behavior of the model softer than that of the actual material. Some judgment is, therefore, required to decide whether or not the solution is sufficiently accurate, or whether the problem can be modeled at all with Abaqus/Explicit because of this numerical limitation. We can assess the relative compressibility of a material by the ratio of its initial bulk modulus, K_0, to its initial shear modulus, μ_0. Poisson's ratio, ν, also provides a measure of compressibility since it is defined as

$$\nu = \frac{3\left(K_0/\mu_0\right) - 2}{6\left(K_0/\mu_0\right) + 2}.$$

Table 1 provides some representative values.

Table 1: Relationship between compressibility and Poisson's ratio.

K_0/μ_0	Poisson's ratio
10	0.452
20	0.475
50	0.490
100	0.495
1,000	0.4995
10,000	0.49995

If no value is given for the material compressibility, by default Abaqus/Explicit assumes $K_0/\mu_0 = 20$, corresponding to Poisson's ratio of 0.475. Since typical unfilled elastomers have K_0/μ_0 ratios in the range of 1,000 to 10,000 ($\nu = 0.4995$ to $\nu = 0.49995$) and filled elastomers have K_0/μ_0 ratios in the range of 50 to 200 ($\nu = 0.490$ to $\nu = 0.497$), this default provides much more compressibility than is available in most elastomers. However, if the elastomer is relatively unconfined, this softer modeling of the material's bulk behavior usually provides quite accurate results. Unfortunately, in cases where the material is highly confined—such as when it is in contact with stiff, metal parts and has a very small amount of free surface, especially when the loading is highly compressive—it may not be feasible to obtain accurate results with Abaqus/Explicit.

If you are defining the compressibility rather than accepting the default value in Abaqus/Explicit, an upper limit of 100 is suggested for the ratio of K_0/μ_0. Larger ratios introduce high frequency noise into the dynamic solution and require the use of excessively small time increments.

Strain energy potential

Abaqus uses a *strain energy potential* (*U*), rather than a Young's modulus and Poisson's ratio, to relate stresses to strains in hyperelastic materials. Several different strain energy potentials are available: a polynomial model, the Ogden model, the Arruda-Boyce model, the Marlow model, and the van der Waals model. Simpler forms of the polynomial model are also available, including the Mooney-Rivlin, neo-Hookean, reduced polynomial, and Yeoh models.

The polynomial form of the strain energy potential is one that is commonly used. Its form is

$$U = \sum_{i+j=1}^{N} C_{ij}\left(\bar{I}_1 - 3\right)^i \left(\bar{I}_2 - 3\right)^j + \sum_{i=1}^{N} \frac{1}{D_i}(J_{el} - 1)^{2i},$$

where *U* is the strain energy potential; J_{el} is the elastic volume ratio; $\bar{I}_1$ and $\bar{I}_2$ are measures of the distortion in the material; and *N*, C_{ij}, and D_i are material parameters, which may be functions of temperature. The C_{ij} parameters describe the shear behavior of the material, and the D_i parameters introduce compressibility. If the material is fully incompressible (a condition not allowed in Abaqus/Explicit), all the values of D_i are set to zero and the second part of the equation shown above can be ignored. If the number of terms, *N*, is one, the initial shear modulus, μ_0, and bulk modulus, K_0, are given by

$$\mu_0 = 2\left(C_{01} + C_{10}\right),$$

and

$$K_0 = \frac{2}{D_1}.$$

If the material is also incompressible, the equation for the strain energy density is

$$U = C_{10}\left(\bar{I}_1 - 3\right) + C_{01}\left(\bar{I}_2 - 3\right).$$

This expression is commonly referred to as the *Mooney-Rivlin* material model. If C_{01} is also zero, the material is called *neo-Hookean*.

The other hyperelastic models are similar in concept and are described in *Hyperelasticity*.

You must provide Abaqus with the relevant material parameters to use a hyperelastic material. For the polynomial form these are C_{ij} and D_i. It is possible that you will be supplied with these parameters when modeling hyperelastic materials; however, more likely you will be given test data for the materials that you must model. Fortunately, Abaqus can accept test data directly and calculate the material parameters for you (using a least squares fit).

Defining hyperelastic behavior using test data

A convenient way of defining a hyperelastic material is to supply Abaqus with experimental test data. Abaqus then calculates the constants using a least squares method. Abaqus can fit data for the following experimental tests:

- Uniaxial tension and compression
- Equibiaxial tension and compression
- Planar tension and compression (pure shear)
- Volumetric tension and compression

The deformation modes seen in these tests are shown in *Figure 1*.

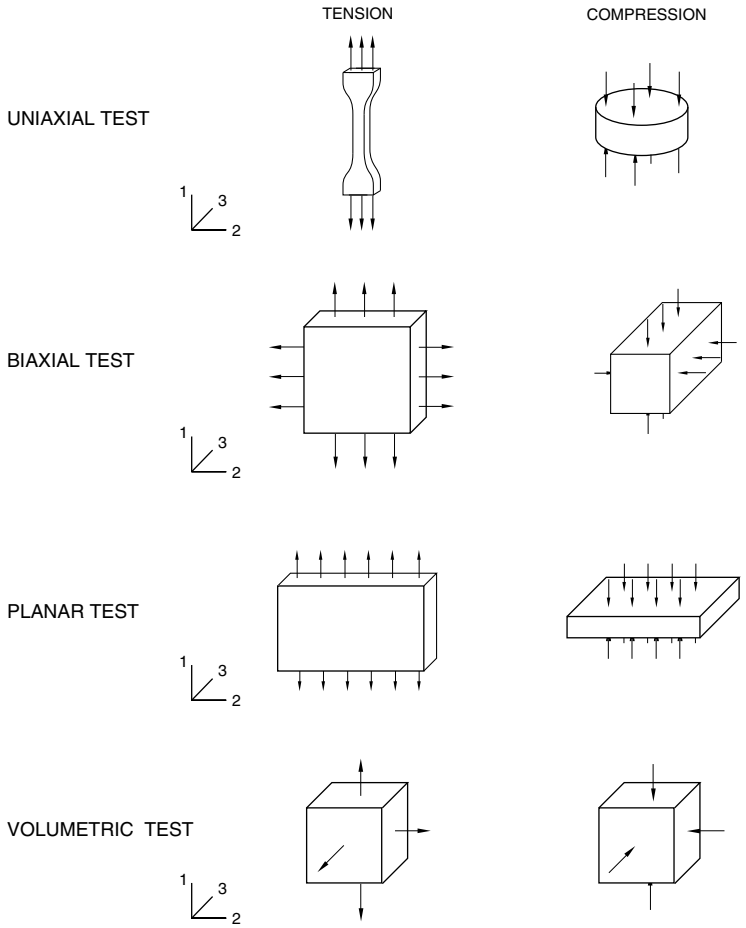


Figure 1: Deformation modes for the various experimental tests for defining hyperelastic material behavior.

Unlike plasticity data, the test data for hyperelastic materials must be given to Abaqus as nominal stress and nominal strain values.

Volumetric compression data only need to be given if the material's compressibility is important. Normally in Abaqus/Standard it is not important, and the default fully incompressible behavior is used. As noted earlier, Abaqus/Explicit assumes a small amount of compressibility if no volumetric test data are given.

Obtaining the best material model from your data

The quality of the results from a simulation using hyperelastic materials strongly depends on the material test data that you provide Abaqus. Typical tests are shown in *Figure 1*. There are several things that you can do to help Abaqus calculate the best possible material parameters.

Wherever possible, try to obtain experimental test data from more than one deformation state—this allows Abaqus to form a more accurate and stable material model. However, some of the tests shown in *Figure 1* produce equivalent deformation modes for incompressible materials. The following are equivalent tests for incompressible materials:

- Uniaxial tension ↔ Equibiaxial compression
- Uniaxial compression ↔ Equibiaxial tension
- Planar tension ↔ Planar compression

You do not need to include data from a particular test if you already have data from another test that models a particular deformation mode.

In addition, the following may improve your hyperelastic material model:

- Obtain test data for the deformation modes that are likely to occur in your simulation. For example, if your component is loaded in compression, make sure that your test data include compressive, rather than tensile, loading.
- Both tension and compression data are allowed, with compressive stresses and strains entered as negative values. If possible, use compression or tension data depending on the application, since the fit of a single material model to both tensile and compressive data will normally be less accurate than for each individual test.
- Try to include test data from the planar test. This test measures shear behavior, which can be very important.
- Provide more data at the strain magnitudes that you expect the material will be subjected to during the simulation. For example, if the material will only have small tensile strains, say under 50%, do not provide much, if any, test data at high strain values (over 100%).
- Use the material evaluation functionality in Abaqus/CAE to perform simulations of the experimental tests and to compare the results Abaqus calculates to the experimental data. If the computational results are poor for a particular deformation mode that is important to you, try to obtain more experimental data for that deformation mode. The technique is illustrated as part of the example discussed in *Example: axisymmetric mount*. Please consult the *Abaqus/CAE User's Guide* for further details.

Stability of the material model

It is common for the hyperelastic material model determined from the test data to be unstable at certain strain magnitudes. Abaqus performs a stability check to determine the strain magnitudes where unstable behavior will occur and prints a warning message in the data (`.dat`) file. The same information is printed in the **Material Parameters and Stability Limit Information** dialog box that appears when the material evaluation capability is used in Abaqus/CAE. You should check this information carefully since your simulation may not be realistic if any part of the model experiences strains beyond the stability limits. The stability checks are done for specific deformations, so it is possible for the material to be unstable below the strain levels indicated if the deformation is more complex. In Abaqus/Standard your simulation may not converge if a part of the model exceeds the stability limits.

See *Hyperelastic Behavior of Rubberlike Materials* for suggestions on improving the accuracy and stability of the test data fit.

Example: axisymmetric mount

The example models the axial stiffness of a rubber mount.

In this section:

- *Axisymmetric mount analysis*
- *Symmetry*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*

Axisymmetric mount analysis

You have been asked to find the axial stiffness of the rubber mount shown below and to identify any areas of high maximum principal stress that might limit the fatigue life of the mount.

The mount is bonded at both ends to steel plates. It will experience axial loads up to 5.5 kN distributed uniformly across the plates. The cross-section geometry and dimensions are given in *Figure 1*.

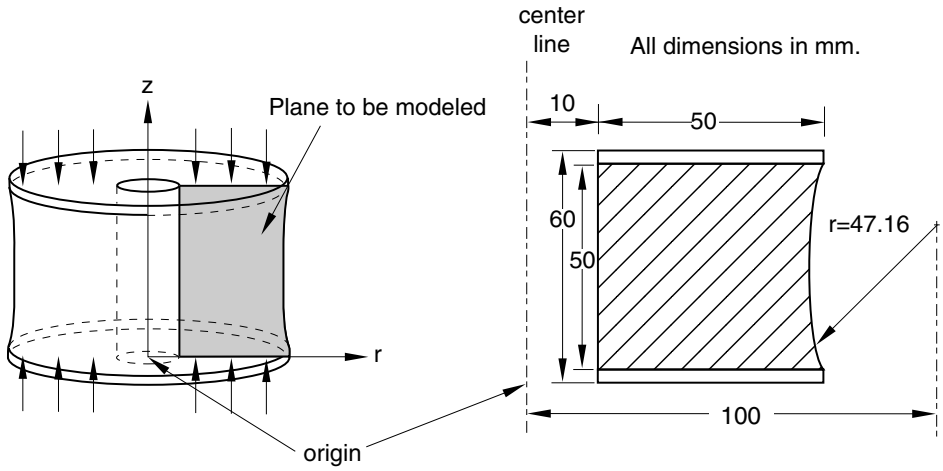


Figure 1: Axisymmetric mount.

You can use axisymmetric elements for this simulation since both the geometry of the structure and the loading are axisymmetric. Therefore, you only need to model a plane through the component: each element represents a complete 360° ring. You will examine the static response of the mount; therefore, you will use Abaqus/Standard for your analysis.

Symmetry

You do not need to model the whole section of this axisymmetric component because the problem is symmetric about a horizontal line through the center of the mount. By modeling only half of the section, you can use half as many elements and, hence, approximately half the number of degrees of freedom. This significantly reduces the run time and storage requirements for the analysis or, alternatively, allows you to use a more refined mesh.

Many problems contain some degree of symmetry. For example, mirror symmetry, cyclic symmetry, axisymmetry, or repetitive symmetry (shown in *Figure 1*) are common. More than one type of symmetry may be present in the structure or component that you want to model.

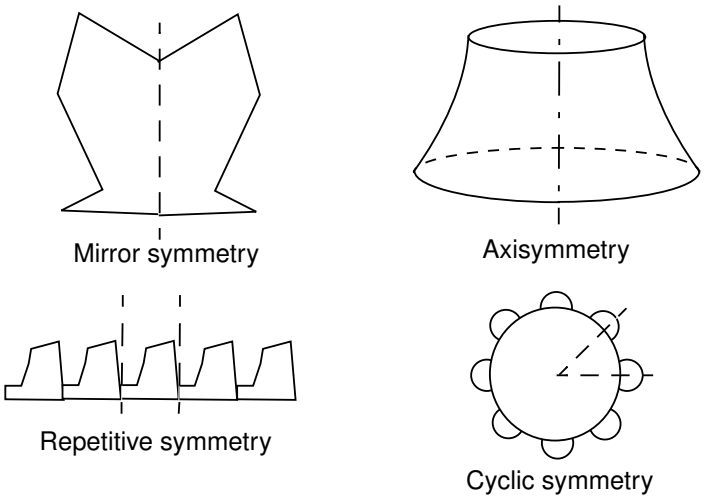


Figure 1: Various forms of symmetry.

When modeling just a portion of a symmetric component, you have to add boundary conditions to make the model behave as if the whole component were being modeled. You may also have to adjust the applied loads to reflect the portion of the structure actually being modeled. Consider the portal frame in *Figure 2*.

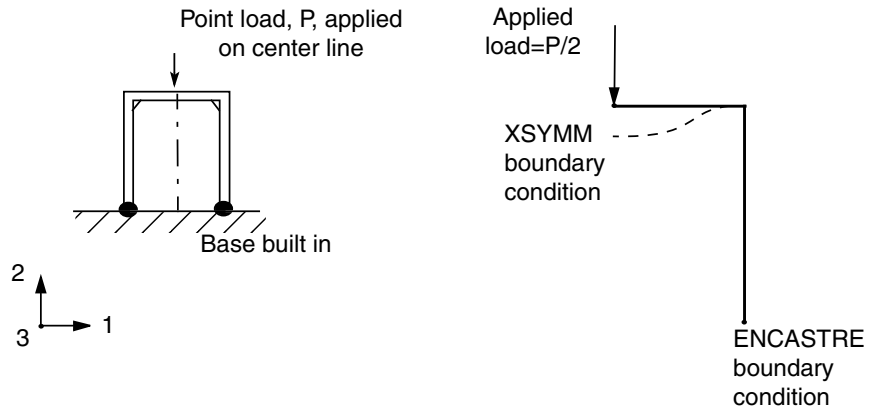


Figure 2: Symmetric portal frame.

The frame is symmetric about the vertical line shown in the figure. To maintain symmetry in the model, any nodes on the symmetry line must be constrained from translating in the 1-direction and from rotating about the 2- or 3-axes.

In the frame problem the load is applied along the model's symmetry plane; therefore, only half of the total value should be applied to the portion you are modeling.

In axisymmetric analyses using axisymmetric elements, such as this rubber mount example, we need model only the cross-section of the component. The element formulation automatically includes the effects of axial symmetry.

Preprocessing—creating the model with Abaqus/CAE

Use Abaqus/CAE to create the model. Abaqus provides scripts that replicate the complete analysis model for this problem.
Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Axisymmetric mount*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Axisymmetric mount**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

In this section:

- *Part definition*
- *Material properties: hyperelastic model for the rubber*
- *Completing the material and section definitions and assigning section properties*
- *Creating an assembly and a step definition*
- *Applying loads and boundary conditions*
- *Creating the mesh and the job*

Part definition

Create an axisymmetric, deformable, shell part. Name the part `Mount`, and specify an approximate part size of `0.3`. Because of symmetry considerations, only the bottom half of the mount will be modeled. You can use the following suggested approach to create the part geometry. When you first enter the sketcher, the axis of revolution is displayed as a green dashed line with a fixed position constraint; your sketch cannot cross this axis.

1. Draw an arbitrary rectangle to the right of the axis of symmetry. Dimension it as follows:
   a. Set the horizontal distance between the axis of symmetry and the left edge of the rectangle to 0.01 m.
   b. Set the vertical height of the rectangle to 0.030 m and its horizontal span to 0.050 m.

2. Sketch a circle using the **Create Circle: Center and Perimeter** tool . Select any point to the right of the rectangle as the center, and snap the perimeter point to any point along the right edge of the rectangle. Dimension the circle as follows:
   a. Set the horizontal distance between the axis of symmetry and the center of the circle to 0.1 m.
   b. Set the vertical distance between the center of the circle and the top-right vertex of the rectangle to 0 m.
   c. Set the vertical distance between the perimeter point (where the circle snaps to the rectangle) and the bottom-right vertex of the rectangle to 0.005 m.

   The sketch appears as shown in *Figure 1*.

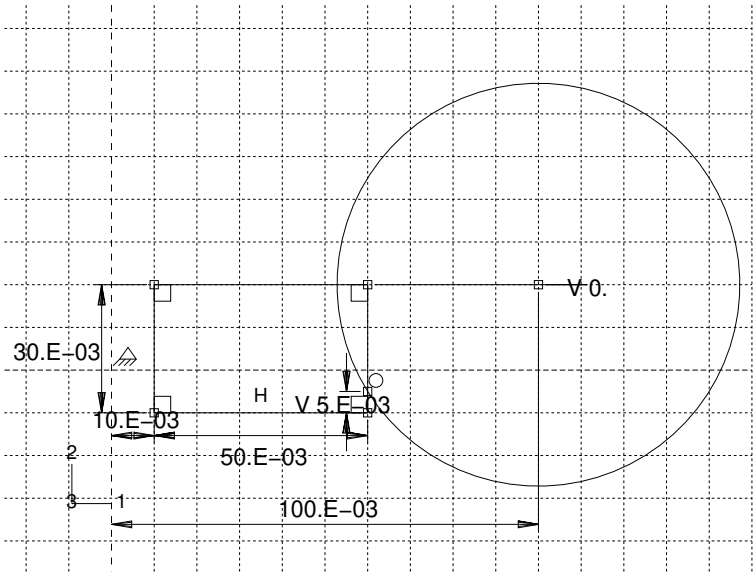


Figure 1: Construction geometry used to create the part (with grid spacing doubled).

3. Use the **Auto-Trim** tool  to remove the excess portions of the sketch as shown in *Figure 2*.

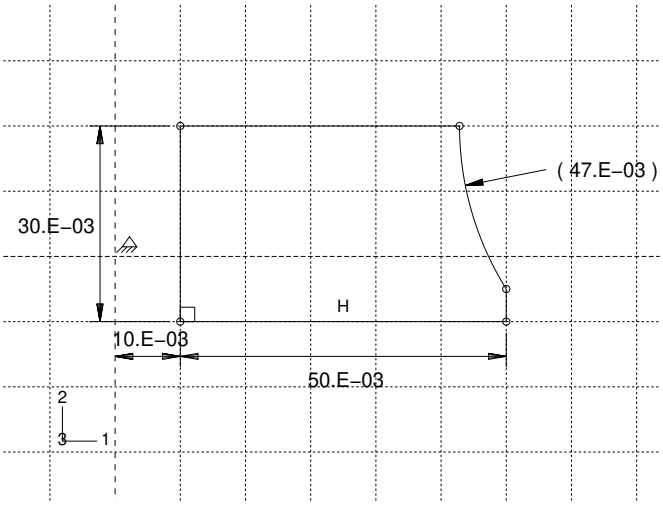


Figure 2: Final part geometry (with grid spacing doubled).

Material properties: hyperelastic model for the rubber

You have been given some experimental test data for the rubber material used in the mount.
Three different sets of test data—a uniaxial test, a biaxial test, and a planar (shear) test—are available.

The data (shown in *Figure 1* and tabulated in *Table 1*, *Table 2*, and *Table 3*) are given in terms of nominal stress and corresponding nominal strain.

Note: Volumetric test data are not required when the material is incompressible (as is the case in this example).

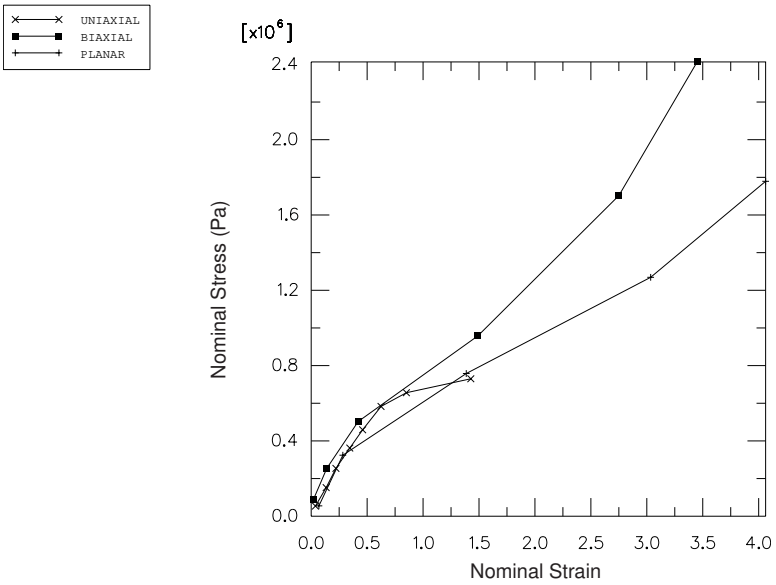


Figure 1: Material test data for the rubber material.

Table 1: Uniaxial test data.

Stress (Pa)	Strain
0.054E6	0.0380
0.152E6	0.1338
0.254E6	0.2210
0.362E6	0.3450
0.459E6	0.4600
0.583E6	0.6242
0.656E6	0.8510
0.730E6	1.4268

Table 2: Biaxial test data.

Stress (Pa)	Strain
0.089E6	0.0200

Stress (Pa)	Strain
0.255E6	0.1400
0.503E6	0.4200
0.958E6	1.4900
1.703E6	2.7500
2.413E6	3.4500

Table 3: Planar test data.

Stress (Pa)	Strain
0.055E6	0.0690
0.324E6	0.2828
0.758E6	1.3862
1.269E6	3.0345
1.779E6	4.0621

When you define a hyperelastic material using experimental data, you also specify the strain energy potential that you want to apply to the data. Abaqus uses the experimental data to calculate the coefficients necessary for the specified strain energy potential. However, it is important to verify that an acceptable correlation exists between the behavior predicted by the material definition and the experimental data.

You can use the material evaluation option available in Abaqus/CAE to simulate one or more standard tests with the experimental data using the strain energy potential that you specify in the material definition.

To define and evaluate hyperelastic material behavior:

1. Create a hyperelastic material named `Rubber`. In this example a first-order, polynomial strain energy function is used to model the rubber material; thus, select **Polynomial** from the **Strain energy potential** list in the material editor. Enter the test data given above using the **Test Data** menu items in the material editor.

   To visualize the experimental data, click mouse button 3 on the table for any of the test data and select **Create *X–Y* Data** from the menu that appears. You can then plot the data in the Visualization module.

   Note: In general, you may be unsure of which strain energy potential to specify. In this case, you could select **Unknown** from the **Strain energy potential** list in the material editor. You could then use the **Evaluate** option to guide your selection by performing standard tests with the experimental data using multiple strain energy potentials.

2. In the Model Tree, click mouse button 3 on **Rubber** underneath the **Materials** container. Select **Evaluate** from the menu that appears to perform the standard unit-element tests (uniaxial, biaxial, and planar). Specify a minimum strain of `0` and a maximum strain of `1.75` for each test. Evaluate only the first-order polynomial strain energy function. This form of the hyperelasticity model is known as the Mooney-Rivlin material model.

   When the evaluation is complete, Abaqus/CAE enters the Visualization module. A dialog box appears containing material parameter and stability information. In addition, an *X–Y* plot that displays a nominal stress–nominal strain curve for the material as well as a plot of the experimental data appears for each test.

The computational and experimental results for the various types of tests are compared in *Figure 2*, *Figure 3*, and *Figure 4* (for clarity, some of the computational data points are not shown). The Abaqus/Standard and experimental results for the biaxial tension test match very well. The computational and experimental results for the uniaxial tension and planar tests match well at strains less than 100%. The hyperelastic material model created from these material test data is probably not suitable for use in general simulations where the strains may be larger than 100%. However, the model will be adequate for this simulation if the principal strains remain within the strain magnitudes where the data and the hyperelastic model fit well. If you find that the results are beyond these magnitudes or if you are asked to perform a different simulation, you will have to insist on getting better material data. Otherwise, you will not be able to have much confidence in your results.

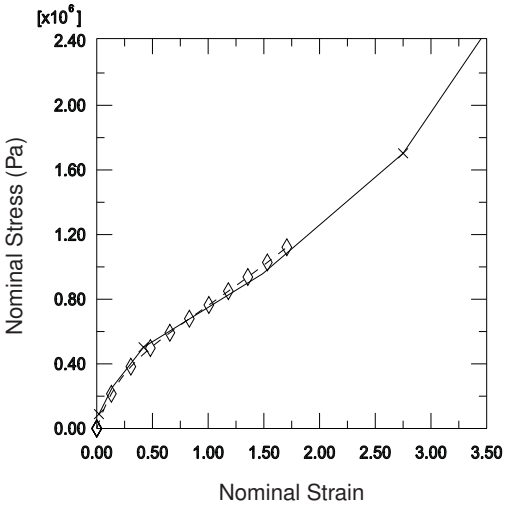


Figure 2: Comparison of experimental data (solid line) and Abaqus/Standard results (dashed line): biaxial tension.

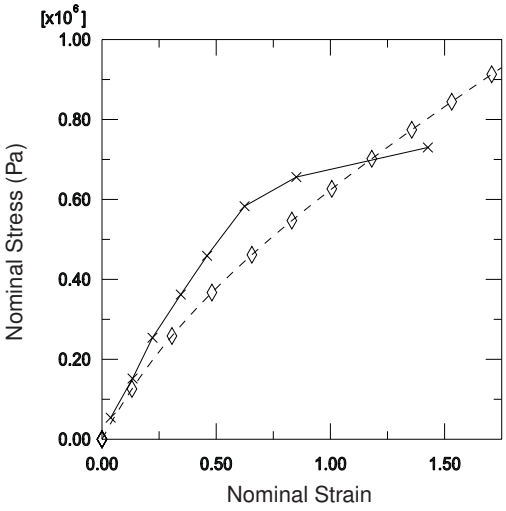


Figure 3: Comparison of experimental data (solid line) and Abaqus/Standard results (dashed line): uniaxial tension.

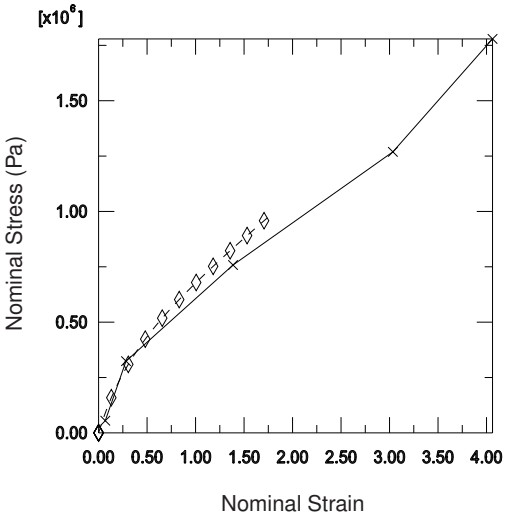


Figure 4: Comparison of experimental data (solid line) and Abaqus/Standard results (dashed line): planar shear.

The hyperelastic material parameters

In this simulation the material is assumed to be incompressible ($D_1 = 0$). To achieve this, no volumetric test data were provided. To simulate compressible behavior, you must provide volumetric test data in addition to the other test data.

The hyperelastic material coefficients—C_{10}, C_{01}, and D_1—that Abaqus calculates from the material test data appear in the **Material Parameters and Stability Limit Information** dialog box, shown in *Figure 5*. The material model is stable at all strains with these material test data and this strain energy function.

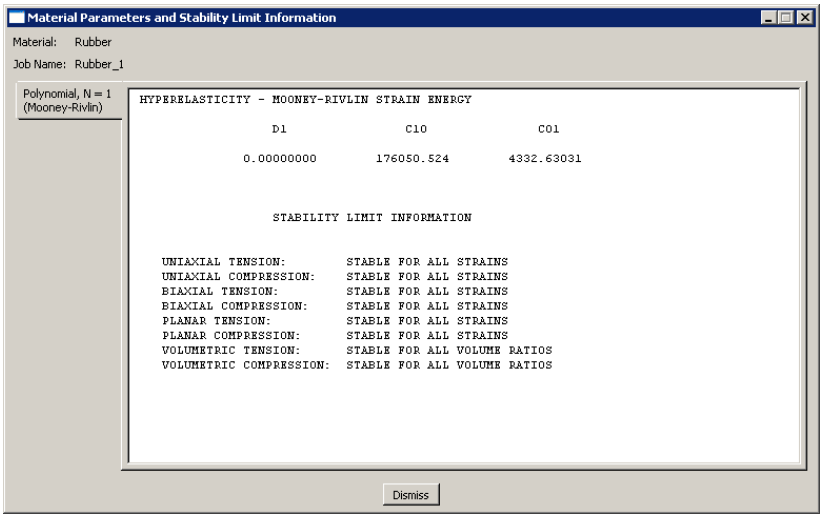


Figure 5: Material parameters and stability limits for the first-order polynomial strain energy function.

However, if you specified that a second-order (N=2) polynomial strain energy function be used, you would see the warnings shown in *Figure 6*. If you had only uniaxial test data for this problem, you would find that the Mooney-Rivlin material model Abaqus creates would have unstable material behavior above certain strain magnitudes.

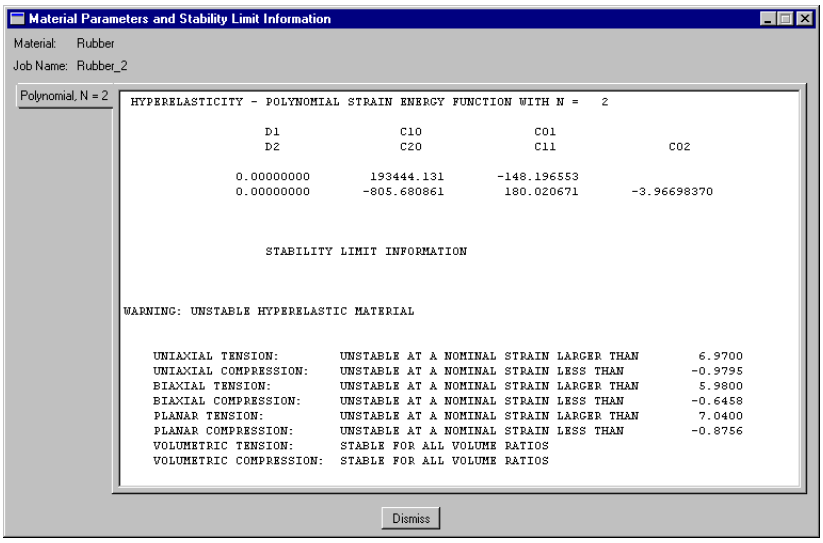


Figure 6: Material parameters and stability limits for the second-order polynomial strain energy function.

Completing the material and section definitions and assigning section properties

The steel is modeled with linear elastic properties only ($E = 200 \times 10^9$ Pa, $\nu = 0.3$) because the loads should not be large enough to cause inelastic deformations.

Create a material named `Steel` with these properties. In addition, create two homogeneous solid section definitions: one named `RubberSection` that refers to the rubber material and one named `SteelSection` that refers to the steel material.

Before assigning section properties, partition the part into the two regions shown in *Figure 1* using the **Partition Face: Sketch** tool.

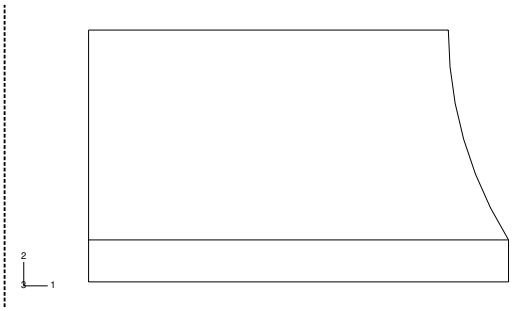


Figure 1: Partition used to divide the part into two regions.

In the Sketcher, draw a horizontal line extending from the point where the circular arc intersects the right edge to any point past the left edge of the sketch.

The upper region represents the rubber mount, while the lower region represents the steel plate. Assign the appropriate section definitions to each region.

Creating an assembly and a step definition

Create a dependent instance of the part.
In this simulation you can accept the default *r*–*z* (1–2) axisymmetric coordinate system.

Then, define a single static, general step named `Compress mount`. When hyperelastic materials are used in a model, Abaqus assumes that the model may undergo large deformations; but large deformations and other nonlinear geometric effects are not included by default in Abaqus/Standard. Therefore, you must include them in this simulation by toggling on **Nlgeom**; otherwise, Abaqus/Standard will terminate the analysis with an input error. Set the time period to `1.0` and the initial time increment to `0.01` (i.e., 1% of the total step time).

For the purpose of restricting output, create a geometry set named `Out` at the vertex located at the lower-left corner of the steel plate region.

Write the preselected variables and nominal strains as field output to the output database file every increment. In addition, write the displacements at a single point on the bottom of the steel plate to the output database file as history data so that the stiffness of the mount can be calculated. Use the geometry set `Out` for this purpose.

Applying loads and boundary conditions

Specify boundary conditions for the region on the symmetry plane (U2 = 0 is shown in *Figure 1*; however, YSYMM would be equivalent in this case).

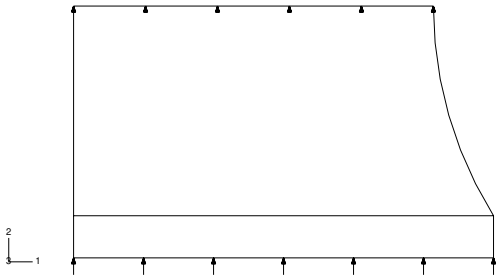


Figure 1: Boundary conditions on the rubber mount; pressure loading on the steel plate.

No boundary constraints are needed in the radial direction (global 1-direction) because the axisymmetric nature of the model does not allow the structure to move as a rigid body in the radial direction. Abaqus will allow nodes to move in the radial direction, even those initially on the axis of symmetry (i.e., those with a radial coordinate of 0.0), if no boundary conditions are applied to their radial displacements (degree of freedom 1). Since you want to let the mount deform radially in this analysis, do not apply any radial boundary conditions; again, Abaqus will prevent rigid body motions automatically.

The mount must carry a maximum axial load of 5.5 kN, spread uniformly over the steel plates. Therefore, apply a distributed load to the bottom of the steel plate, as shown in *Figure 1*. The magnitude of the pressure is given by

$$p = 5500/\left(\pi\left(0.06^2 - 0.01^2\right)\right) \cong 0.50 \times 10^6 \text{ Pa}.$$

Creating the mesh and the job

Use first-order, axisymmetric, hybrid solid elements (CAX4H) for the rubber mount. You must use hybrid elements because the material is fully incompressible. The elements are not expected to be subjected to bending, so shear locking in these fully integrated elements should not be a concern. Model the steel plates with a single layer of incompatible mode elements (CAX4I) because it is possible that the plates may bend as the rubber underneath them deforms.

Create a quadrilateral mesh. Seed the part by specifying the number of elements along the edges (by selecting **Seed->Edges** or the  tool). Specify `30` elements along each horizontal edge, `14` elements along the vertical and curved edges of the rubber, and `1` element along the vertical edges of the steel. The mesh is shown in *Figure 1*.

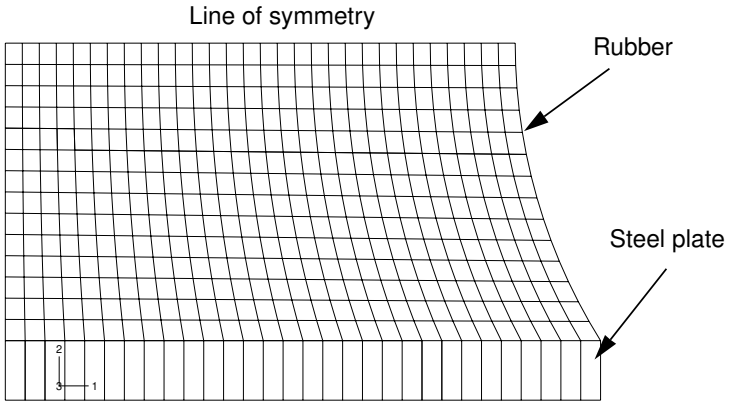


Figure 1: Mesh for the rubber mount.

Create a job named `Mount`. Give the job the following description: `Axisymmetric mount analysis under axial loading`.

Save your model in a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate and correct as necessary the cause of any warning messages.

Postprocessing

Enter the Visualization module, and open the file `Mount.odb`.

Calculating the stiffness of the mount

Determine the stiffness of the mount by creating an *X*–*Y* plot of the displacement of the steel plate as a function of the applied load. You will first create a plot of the vertical displacement of the node on the steel plate for which you wrote data to the output database file. Data were written for the node in set `Out` in this model.

To create a history curve of vertical displacement and swap the *X*- and *Y*-axes:

1. In the Results Tree, expand the **History Output** container underneath the output database named `Mount.odb`.
2. Locate and select the vertical displacement U2 at the node in set `Out`.
3. Click mouse button 3, and select **Save As** from the menu that appears to save the *X*–*Y* data.

   The **Save XY Data As** dialog box appears.
4. In the **Save XY Data As** dialog box, name the curve `SWAPPED` and select **swap(XY)** as the save operation; click **OK**.

   The plot of time-displacement appears in the viewport.

You now have a curve of time-displacement. What you need is a curve showing force-displacement. This is easy to create because in this simulation the force applied to the mount is directly proportional to the total time in the analysis. All you have to do to plot a force-displacement curve is multiply the curve `SWAPPED` by the magnitude of the load (5.5 kN).

To multiply a curve by a constant value:

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.
2. Select **Operate on XY data**, and click **Continue**.

   The **Operate on XY Data** dialog box appears.
3. In the **XY Data** field, double-click `SWAPPED`.

   The expression `"SWAPPED"` appears in the text field at the top of the dialog box. Your cursor should be at the end of the text field.
4. Multiply the data object in the text field by the magnitude of the applied load by entering `*5500`.
5. Save the multiplied data object by clicking **Save As** at the bottom of the dialog box.

   The **Save XY Data As** dialog box appears.
6. In the **Name** text field, type `FORCEDEF`; and click **OK** to close the dialog box.
7. To view the force-displacement plot, click **Plot Expression** at the bottom of the **Operate on XY Data** dialog box.

You have now created a curve with the force-deflection characteristic of the mount (the axis labels do not reflect this since you did not change the actual variable plotted). To get the stiffness, you need to differentiate the curve `FORCEDEF`. You can do this by using the `differentiate( )` operator in the **Operate on XY Data** dialog box.

To obtain the stiffness:

1. In the **Operate on XY Data** dialog box, clear the current expression.
2. From the **Operators** listed, click **differentiate(X)**.

   `differentiate( )` appears in the text field at the top of the dialog box.
3. In the **XY Data** field, double-click `FORCEDEF`.

   The expression `differentiate( "FORCEDEF" )` appears in the text field.
4. Save the differentiated data object by clicking **Save As** at the bottom of the dialog box.

   The **Save XY Data As** dialog box appears.
5. In the **Name** text field, type `STIFF`; and click **OK** to close the dialog box.
6. To plot the stiffness-displacement curve, click **Plot Expression** at the bottom of the **Operate on XY Data** dialog box.
7. Click **Cancel** to close the dialog box.
8. Open the **Axis Options** dialog box, and switch to the **Title** tabbed page.
9. Customize the axis titles so they appear as shown in *Figure 1*.

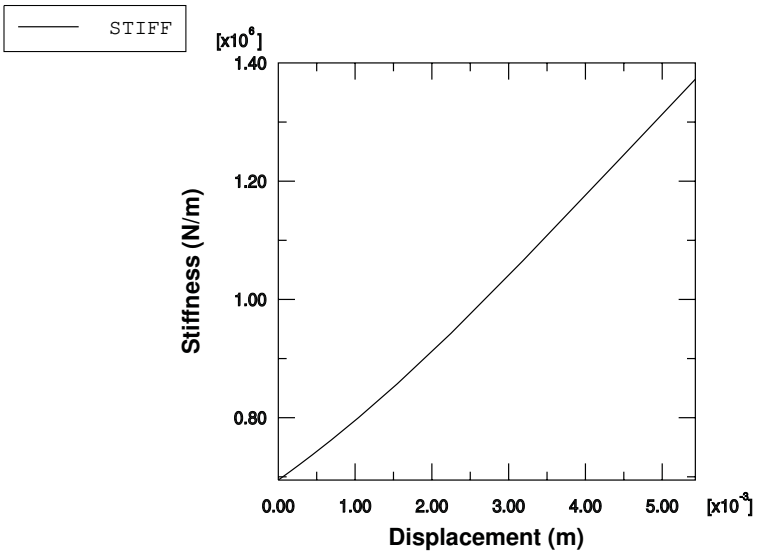


Figure 1: Stiffness characteristic of the mount.

10. Click **Dismiss** to close the **Axis Options** dialog box.

The stiffness of the mount increases by almost 100% as the mount deforms. This is a result of the nonlinear nature of the rubber and the change in shape of the mount as it deforms. Alternatively, you could have created the stiffness-displacement curve directly by combining all the operators above into one expression.

To define the stiffness curve directly:

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.
2. Select **Operate on XY data**, and click **Continue**.

   The **Operate on XY Data** dialog box appears.

3. From the **Operators** listed, click **differentiate(X)**.

   `differentiate( )` appears in the text field at the top of the dialog box.

4. In the **XY Data** field, double-click `SWAPPED`.

   The expression `differentiate( "SWAPPED" )` appears in the text field.

5. Place the cursor in the text field directly after the `"SWAPPED"` data object, and type `*5500` to multiply the swapped data by the constant total force value.

   `differentiate( "SWAPPED"*5500 )` appears in the text field.

6. Save the differentiated data object by clicking **Save As** at the bottom of the dialog box.

   The **Save XY Data As** dialog box appears.

7. In the **Name** text field, type `STIFFNESS`; and click **OK** to close the dialog box.

8. Click **Cancel** to close the **Operate on XY Data** dialog box.

9. Customize the *X*- and *Y*-axis labels as they appear in *Figure 1* if you have not already done so.

10. In the Results Tree, click mouse button 3 on `STIFFNESS` underneath the **XYData** container and select **Plot** from the menu that appears to view the plot in *Figure 1* that shows the variation of the mount's axial stiffness as the mount deforms.

Model shape plots

You will begin by plotting the undeformed model shape of the mount.

To plot the undeformed model shape:

From the main menu bar, select **Plot->Undeformed Shape**; or use the  tool in the Visualization module toolbox to plot the undeformed model shape (see *Figure 2*).

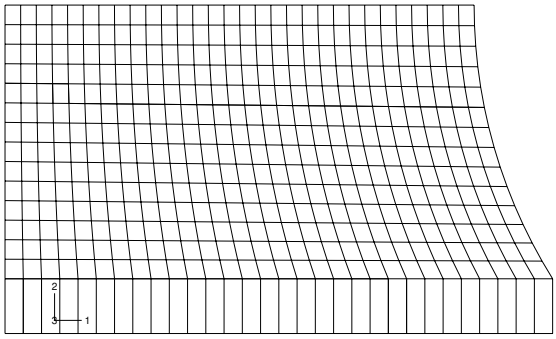


Figure 2: Undeformed model shape of the rubber mount.

If the figure obscures the plot title, you can move the plot by clicking the  tool and holding down mouse button 1 to pan the deformed shape to the desired location. Alternatively, you can turn the plot title off (**Viewport->Viewport Annotation Options**).

In this figure the axisymmetric model is displayed as a planar, two-dimensional shape. You can producing a three-dimensional visual effect by sweeping the model through a specified angle. In addition, you can also mirror results about selected planes (such as symmetry planes) to render results on a full three-dimensional representation of the model. These are visualization aids only. Any numerical representation of the results, such as the contour

legend, indicates only the portion of the model that was analyzed. Since in this problem the symmetry plane does not necessarily coincide with one of the global coordinate system planes, a local system will be defined to facilitate the mirroring operation.

To define a local coordinate system for postprocessing:

1. From the main menu bar, select **Tools->Coordinate System->Create**.
2. In the **Create Coordinate System** dialog box, enter `rectangular` as the name and click **Continue**.
3. Select the node at the top-left corner of the model as the origin, the node at the top-right corner as the point on the *X*-axis, and the node at the bottom-left corner as the point in the *X*–*Y* plane.

To mirror and sweep the cross-section:

1. From the main menu bar, select **View->ODB Display Options**.
2. In the **ODB Display Options** dialog box, click the **Mirror/Pattern** tab.
3. Select **rectangular** from the **Mirror CSYS** list.
4. Select **XZ** as the mirror plane.
5. Click **Apply**.

   The mirrored image appears.
6. Click the **Sweep/Extrude** tab.
7. Toggle on **Sweep elements** and set the sweep range from `0` to `270` degrees. Set the number of segments to `45`.
8. Click **OK**.

   The swept image appears. To more clearly distinguish between the rubber and the steel, color code the model based on section assignment.

You will now plot the deformed model shape of the mount. This will allow you to evaluate the quality of the deformed mesh and to assess the need for mesh refinement.

To plot the deformed model shape:

From the main menu bar, select **Plot->Deformed Shape**, or use the  tool to plot the deformed model shape of the mount (see *Figure 3*).

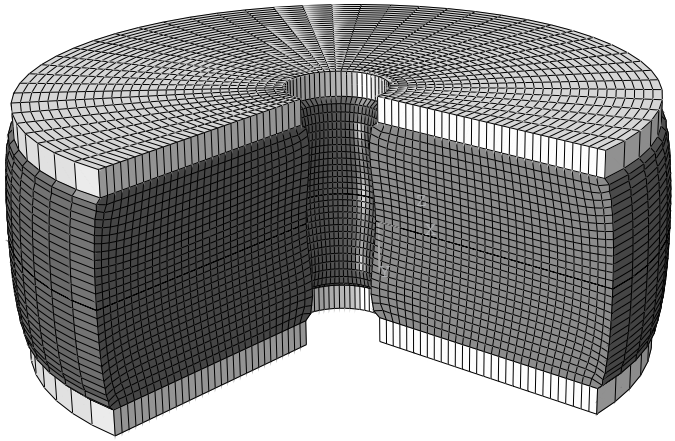


Figure 3: Deformed model shape of the rubber under an applied load of 5500 N (mirrored/swept image).

The plate has been pushed up, causing the rubber to bulge at the sides. Zoom in on the bottom left corner of the mesh using the  tool from the View Manipulation toolbar. Click mouse button 1, and hold it down to define the first corner of the new view; move the mouse to create a box enclosing the viewing area that you want (*Figure 4*); and release the mouse button. Alternatively, you can zoom and pan the plot by selecting **View->Specify** from the main menu bar.

You should have a plot similar to the one shown in *Figure 4* (in this and the images that follow, the sweep and mirroring operations applied earlier have been suppressed).

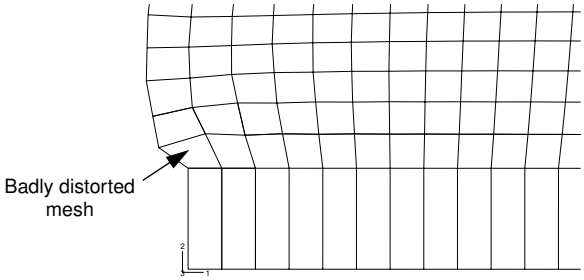


Figure 4: Distortion at the left-hand corner of the rubber mount model.

Some elements in this corner of the model are becoming badly distorted because the mesh design in this area was inadequate for the type of deformation that occurs there. Although the shape of the elements is fine at the start of the analysis, they become badly distorted as the rubber bulges outward, especially the element in the corner. If the loading were increased further, the element distortion may become so excessive that the analysis may exit. *Mesh design for large distortions* discusses how to improve the mesh design for this problem.

The keystoning pattern exhibited by the distorted elements in the bottom right-hand corner of the model indicates that they are locking. A contour plot of the hydrostatic pressure stress in these elements (without averaging across elements sharing common nodes) shows rapid variation in the pressure stress between adjacent elements. This indicates that these elements are suffering from volumetric locking, which was discussed earlier in *Selecting elements for elastic-plastic problems*, in the context of plastic incompressibility. Volumetric locking arises in this problem from overconstraint. The steel is very stiff compared to the rubber. Thus, along the bond line the rubber elements cannot deform laterally. Since these elements must also satisfy incompressibility requirements, they are highly constrained and locking occurs. Analysis techniques that address volumetric locking are discussed in *Techniques for reducing volumetric locking*.

Contouring the maximum principal stress

Plot the maximum in-plane principal stress in the model. Follow the procedure given below to create a filled contour plot on the actual deformed shape of the mount with the plot title suppressed.

To contour the maximum principal stress:

1. By default, Abaqus/CAE displays **S**, **Mises** as the primary field output variable. In the **Field Output** toolbar, select **Max. Principal** as the invariant.

   Abaqus/CAE automatically changes the current plot state to display a contour plot of the maximum in-plane principal stresses on the deformed model shape.
2. Open the **Contour Plot Options** dialog box.
3. Drag the uniform contour intervals slider to `8`.
4. Click **OK** to view the contour plot and to close the dialog box.

   Create a display group showing only the elements in the rubber mount.
5. In the Results Tree, expand the **Materials** container underneath the output database file named `Mount.odb`.
6. Click mouse button 3 on **RUBBER**, and select **Replace** from the menu that appears to replace the current display with the selected elements.
7. The viewport display changes and displays only the rubber mount elements, as shown in *Figure 5*.

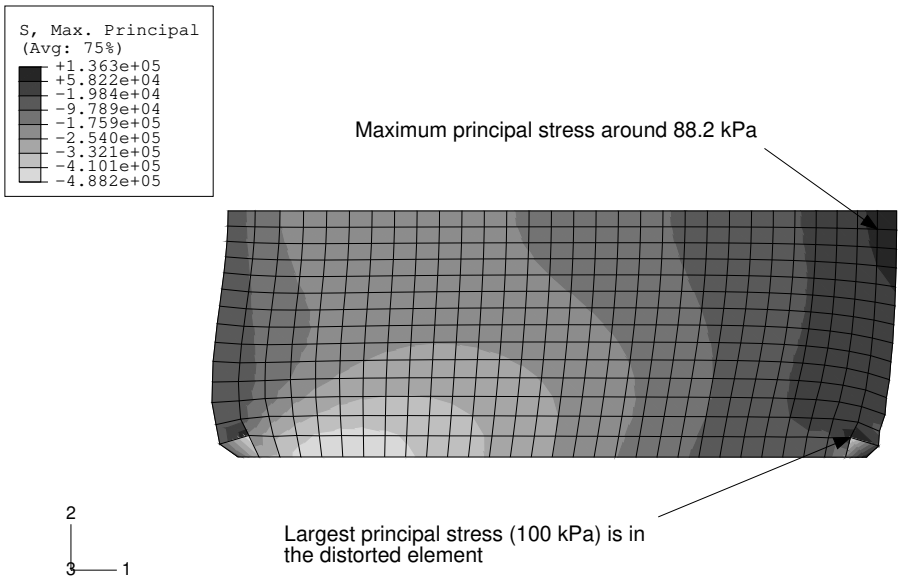


Figure 5: Contours of maximum principal stress in the rubber mount.

The maximum principal stress in the model, reported in the contour legend, is 136 kPa. Although the mesh in this model is fairly refined and, thus, the extrapolation error should be minimal, you may want to use the query tool to determine the more accurate integration point values of the maximum principal stress.

When you look at the integration point values, you will discover that the peak value of maximum principal stress occurs in one of the distorted elements in the bottom right-hand part of the model. This value is likely to be unreliable because of the levels of element distortion and volumetric locking. If this value is ignored, there is an area near the plane of symmetry where the maximum principal stress is around 88.2 kPa.

The easiest way to check the range of the principal strains in the model is to display the maximum and minimum values in the contour legend.

To check the principal nominal strain magnitude:

1. From the main menu bar, select **Viewport->Viewport Annotation Options**.

   The **Viewport Annotation Options** dialog box appears.
2. Click the **Legend** tab, and toggle on **Show min/max values**.
3. Click **OK**.

   The maximum and minimum values appear at the bottom of the contour legend in the viewport.
4. In the **Field Output** toolbar, select **Primary** as the variable type if it is not already selected.

   Abaqus/CAE automatically changes the current plot state to display a contour plot of the maximum in-plane principal stresses on the deformed model shape.
5. From the list of output variables, select **NE**.
6. From the list of invariants in the **Field Output** toolbar, select **Max. Principal** if it is not already selected.

   The contour plot changes to display values for maximum principal nominal strain. Note the value of the maximum principal nominal strain from the contour legend.
7. From the list of invariants, select **Min. Principal**.

   The contour plot changes to display values for minimum principal nominal strain. Note the value of the minimum principal nominal strain from the contour legend.

The maximum and minimum principal nominal strain values indicate that the maximum tensile nominal strain in the model is about 100% and the maximum compressive nominal strain is about 56%. Because the nominal strains in the model remained within the range where the Abaqus hyperelasticity model has a good fit to the material data, you can be fairly confident that the response predicted by the mount is reasonable from a material modeling viewpoint.

Mesh design for large distortions

We know that the element distortions in the corners of the rubber mount are undesirable. The results in these areas are unreliable; and if the load were increased, the analysis might fail. These problems can be corrected by using a better mesh design. The mesh shown in *Figure 1* is an example of an alternate mesh design that might be used to reduce element distortion in the bottom left corner of the rubber model.

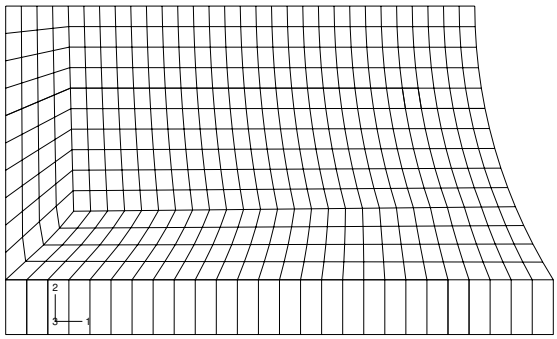


Figure 1: Modified mesh to minimize element distortions in the bottom left corner of the rubber model during the simulation.

The issues surrounding the mesh distortion in the opposite corner are addressed in *Techniques for reducing volumetric locking*. The elements in the bottom left-hand corner region are now much more distorted in the initial, undeformed configuration. However, as the analysis progresses and the elements deform, their shape actually improves. The deformed shape plot, shown in *Figure 2*, illustrates that the amount of element distortion in this region is reduced; however, the level of mesh distortion in the bottom right-hand corner of the rubber model is still significant.

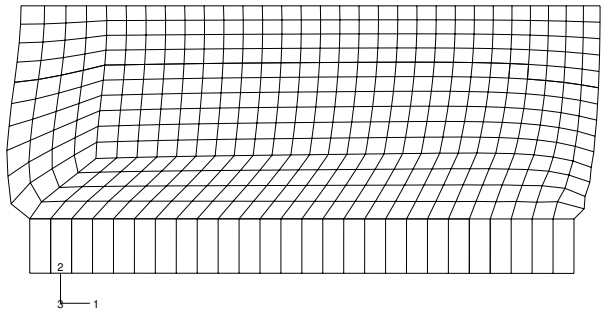


Figure 2: Deformed shape of the modified mesh.

The contours of maximum principal stress (*Figure 3*) show that the very localized stress in that corner has been reduced only slightly.

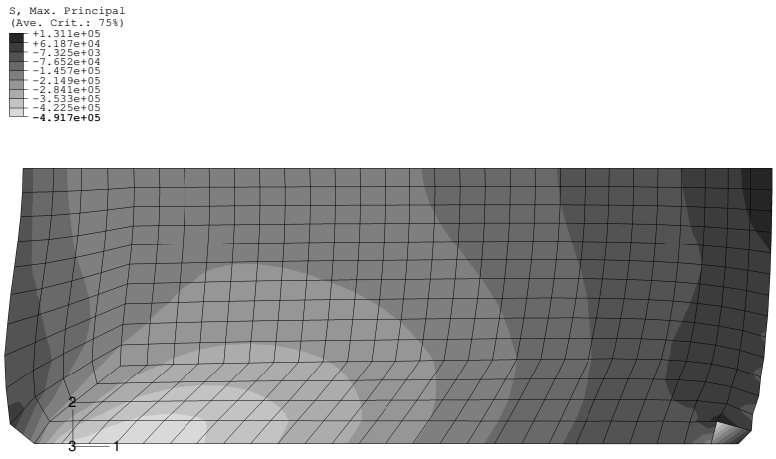


Figure 3: Contours of maximum principal stress in the modified mesh.

Mesh design for large-distortion problems is more difficult than it is for small-displacement problems. A mesh must be produced where the shape of the elements is reasonable throughout the analysis, not just at the start. You must estimate how the model will deform using experience, hand calculations, or the results from a coarse finite element model.

Techniques for reducing volumetric locking

A small amount of compressibility is introduced into the rubber material model in order to alleviate volumetric locking. Provided the amount of compressibility is small, the results obtained with a nearly incompressible material will be very similar to those obtained with an incompressible material.

Compressibility is introduced by setting the material constant D_1 to a nonzero value. The value is chosen so that the initial Poisson's ratio, ν_0, is close to 0.5. The equations given in *Hyperelastic Behavior of Rubberlike Materials* can be used to relate D_1 and ν_0 in terms of μ_0 and K_0 (the initial shear and bulk moduli, respectively) for the polynomial form of the strain energy potential. For example, the hyperelastic material coefficients obtained earlier from the test data (see "The hyperelastic material parameters" in *Preprocessing—creating the model with Abaqus/CAE*) were given as $C_{10} = 176051$ and $C_{01} = 4332.63$; thus, setting $D_1 = 5.E{-}7$ yields $\nu_0 = 0.46$.

A model incorporating compressibility with additional mesh refinement (to reduce mesh distortion) is shown in *Figure 1* (this mesh can be generated easily by changing the edge seeds in Abaqus/CAE or another preprocessor).

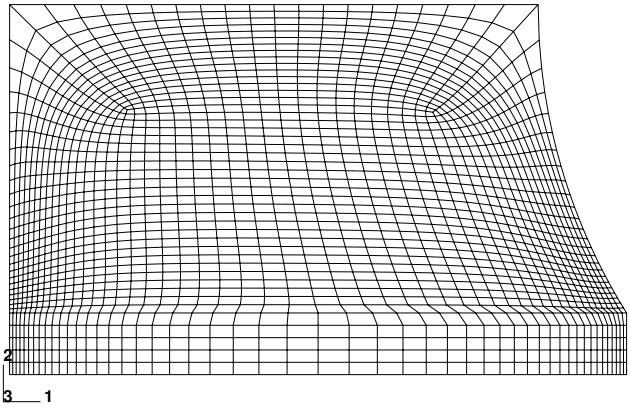


Figure 1: Modified mesh with refinement at both corners.

The deformed shape associated with this model is shown in *Figure 2*.

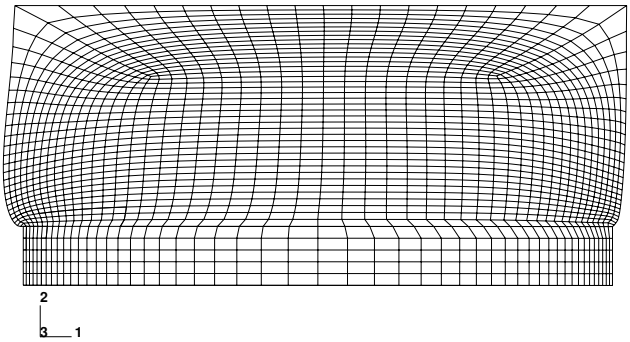


Figure 2: Deformed shape of the modified mesh.

It is clear from this figure that the mesh distortion has been reduced significantly in the critical regions of the rubber model. Examining contour plots of the pressure (without averaging across elements) reveals a smooth variation in pressure stress between elements. Thus, volumetric locking has been eliminated.

Related Abaqus examples

Related Abaqus examples are available.

- *Pressurized rubber disc*
- *Necking of a round tensile bar*
- *Fitting of rubber test data*
- *Uniformly loaded, elastic-plastic plate*

Suggested reading

The following provides the interested user with additional references on material modeling.

General texts on materials

- Ashby, M. F., and D. R. H. Jones, *Engineering Materials*, Pergamon Press, 1980.
- Callister, W. D., *Materials Science & Engineering—An Introduction*, John Wiley, 1994.
- Pascoe, K. J., *An Introduction to the Properties of Engineering Materials*, Van Nostrand, 1978.

Plasticity

- SIMULIA, *Metal Inelasticity in Abaqus*.
- Lubliner, J., *Plasticity Theory*, Macmillan Publishing Co., 1990.
- Calladine, C. R., *Engineering Plasticity*, Pergamon Press, 1969.

Rubber elasticity

- SIMULIA, *Modeling Rubber and Viscoelasticity with Abaqus*.
- Gent, A., *Engineering with Rubber (How to Design Rubber Components)*, Hanser Publishers, 1992.

Summary

This section presents a summary of the concepts.

- Abaqus contains an extensive library to model the behavior of various engineering materials. It includes models for metal plasticity and rubber elasticity.
- The stress-strain data for the metal plasticity model must be defined in terms of true stress and true plastic strain. Nominal stress-strain data can be converted easily into true stress-strain data.
- The metal plasticity model in Abaqus assumes incompressible plastic behavior.
- For efficiency Abaqus/Explicit *regularizes* user-defined material curves by fitting them with curves composed of equally spaced points.
- The hyperelastic material model in Abaqus/Standard allows true incompressibility. The hyperelastic material model in Abaqus/Explicit does not: the default Poisson's ratio for hyperelastic materials in Abaqus/Explicit is 0.475. Some analyses may require increasing Poisson's ratio to model incompressibility more accurately.
- Polynomial, Ogden, Arruda-Boyce, Marlow, van der Waals, Mooney-Rivlin, neo-Hookean, reduced polynomial, and Yeoh strain energy functions are available for rubber elasticity (hyperelasticity). All models allow the material coefficients to be determined directly from experimental test data. The test data must be specified as nominal stress and nominal strain values.
- The material evaluation capability in Abaqus/CAE can be used to verify the correlation between the behavior predicted by the hyperelastic material models and experimental test data.
- Stability warnings may indicate that a hyperelastic material model is unsuitable for the strain ranges you wish to analyze.
- The presence of symmetry can be used to reduce the size of a simulation since only part of the component needs to be modeled. The effect of the rest of the component is represented by applying appropriate boundary conditions.
- Mesh design for large-distortion problems is more difficult than for small-displacement problems. The elements in the mesh must not become too distorted at any stage of the analysis.
- Volumetric locking can be alleviated by permitting a small amount of compressibility. Care must be taken to ensure that the amount of compressibility introduced into the problem does not grossly affect the overall results.
- The *X*–*Y* plotting capabilities in Abaqus/CAE allow data in curves to be manipulated to create new curves. Two curves or a curve and a constant can be added, subtracted, multiplied, or divided. Curves can also be differentiated, integrated, and combined.

Multiple Step Analysis

The general goal of an Abaqus simulation is to predict the response of a structure to applied loads. Recall that in a general sense the term *load* in Abaqus refers to anything that induces a change in the response of a structure from its initial state; for example, nonzero boundary conditions or applied displacements, point forces, pressures, fields, etc. In some cases loads are relatively simple, such as a single set of point loads on a structure. In other problems the loads applied to a structure can be very complex. For example, different loads may be applied to different portions of the model in a particular sequence over some period of time, or the magnitude of the loads may vary as a function of time. The term *load history* is used to refer to such complex loading of a model.

In Abaqus the user divides the complete load history of the simulation into a number of *steps*. Each step is a period of "time," specified by the user, for which Abaqus calculates the response of the model to a particular set of loads and boundary conditions. The user must specify the type of response, known as the analysis procedure, during each step and may change analysis procedures from step to step. For example, static dead loads, perhaps gravitational loads, could be applied to a structure in one step; and the dynamic response of the loaded structure to earthquake accelerations could be calculated in the next step. Both implicit and explicit analyses can contain multiple steps; however, implicit and explicit steps cannot be combined in the same analysis job. To combine a series of implicit and explicit steps, the results transfer (or import) capability can be used. This feature is discussed in *Transferring Results between Abaqus/Explicit and Abaqus/Standard* and is not discussed further here.

Abaqus divides all of its analysis procedures into two main groups: linear perturbation and general. General analysis steps can be included in an Abaqus/Standard or an Abaqus/Explicit analysis; linear perturbation steps are available only in Abaqus/Standard. Loading conditions and "time" are defined differently for the two cases. Furthermore, the results from each type of procedure should be interpreted differently.

The response of the model during a general analysis procedure, known as a *general step*, may be either nonlinear or linear. In a step that uses a perturbation procedure, which is called a *perturbation step*, the response can only be linear. Abaqus/Standard treats such steps as a linear perturbation about the preloaded, predeformed state (known as the base state) created by any previous general steps; therefore, its capability for doing linear simulations is rather more general than that of a purely linear analysis program.

In this section:

- *General analysis procedures*
- *Linear perturbation analysis*
- *Example: vibration of a piping system*
- *Restart analysis*
- *Example: restarting the pipe vibration analysis*
- *Related Abaqus examples*
- *Summary*

General analysis procedures

The starting point for each general step is the end of the last general step.
Therefore, the state of the model evolves in a sequence of general steps as it responds to the loads defined in each step. Any initial conditions define the starting point for the first general step in the simulation.

All general analysis procedures share the same concepts for applying loads and defining “time.”

In this section:

- *Time in general analysis steps*
- *Specifying loads in general steps*

Time in general analysis steps

Abaqus has two measures of time in a simulation. The *total time* increases throughout all general steps and is the accumulation of the time from each general step. Each step also has its own time scale (known as the *step time*), which begins at zero for each step. Time varying loads and boundary conditions can be specified in terms of either time scale. The time scales for an analysis whose history is divided into three steps, each 100 seconds long, are shown in *Figure 1*.

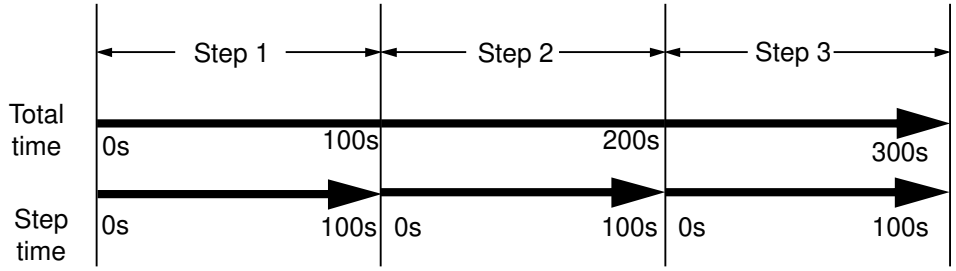


Figure 1: Step and total time for a simulation.

Specifying loads in general steps

In general steps the loads must be specified as total values, not incremental values. For example, if a concentrated load has a value of 1000 N in the first step and it is increased to 3000 N in the second general step, the magnitude given for the load in the two steps should be 1000 N and 3000 N, not 1000 N and 2000 N.

By default, all previously defined loads are propagated to the current step. In the current step you can define additional loads as well as modify any previously defined load (for example, change its magnitude or deactivate it). Any previously defined load that is not specifically modified in the current step continues to follow its associated amplitude definition, provided that the amplitude curve is defined in terms of total time; otherwise, the load is maintained at the magnitude it had at the end of the last general step.

Linear perturbation analysis

This section discusses linear perturbation analysis.

In this section:

- *Linear perturbation steps*
- *Specifying loads in linear perturbation steps*

Linear perturbation steps

Linear perturbation analysis steps are available only in Abaqus/Standard.

The starting point for a linear perturbation step is called the base state of the model. If the first step in a simulation is a linear perturbation step, the base state is the state of the model specified using initial conditions. Otherwise, the base state is the state of the simulation at the end of the last general step prior to the linear perturbation step. Although the response of the structure during the perturbation step is by definition linear, the model may have a nonlinear response in previous general steps. For models with a nonlinear response in the prior general steps, Abaqus/Standard uses the current elastic modulus as the linear stiffness for perturbation procedures. This modulus is the initial elastic modulus for elastic-plastic materials and the tangent modulus for hyperelastic materials (see *Figure 1*); the moduli used for other material models are described in *General and Perturbation Procedures*.

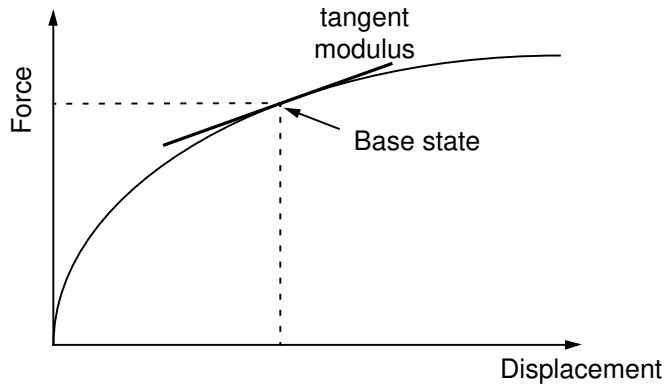


Figure 1: For hyperelastic materials the tangent modulus is used as the stiffness in linear perturbation steps that occur after general, nonlinear steps.

The loads in the perturbation step should be sufficiently small that the model's response would not deviate much from that predicted with the tangent modulus. If the simulation includes contact, by default the contact state between two surfaces does not change during a perturbation step: points that were closed in the base state remain closed, and points that were open remain open. Optionally, contact constraints can be enforced only where the contact pressure in the base state exceeds a specified threshold.

If another general step follows a perturbation step, Abaqus/Standard uses the state of the model at the end of the last general step as its starting point, not the state of the model at the end of the perturbation step. Thus, the response from a linear perturbation step has no permanent effect on the simulation. Therefore, Abaqus/Standard does not include the step time of linear perturbation steps in the total time for the analysis. In fact, what Abaqus/Standard actually does is to define the step time of a perturbation step to be very small (10^{-36}) so that it has no effect when it is added to the total accumulated time. The exception to this rule is the modal dynamics procedure.

Specifying loads in linear perturbation steps

Loads and prescribed boundary conditions given in linear perturbation steps are always local to that step. The load magnitudes (including the magnitudes of prescribed boundary conditions) given in a linear perturbation step are always the perturbation (increment) of the load, not the total magnitude. 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.

As an example of a simple load history that includes a mixture of general and perturbation steps, consider the bow and arrow shown in *Figure 1*.

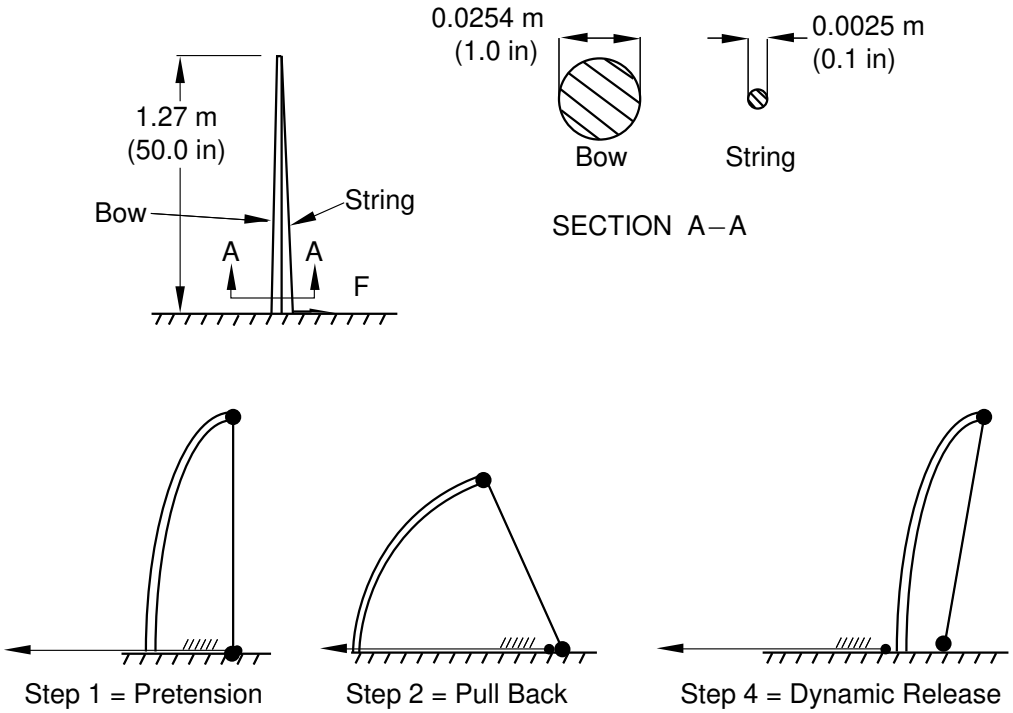


Figure 1: Simple bow and arrow.

Step 1 might be to string the bow—to pretension the bowstring. Step 2 would then follow this by pulling back the string with an arrow, thus storing more strain energy in the system. Step 3 might then be a linear perturbation analysis step: an eigenvalue frequency analysis to investigate the natural frequencies of the loaded system. Such a step might also have been included between Steps 1 and 2, to find the natural frequencies of the bow and string just after the string is pretensioned but before it is pulled back to shoot. Step 4 is then a nonlinear dynamic analysis, in which the bowstring is released, so that the strain energy that was stored in the system by pulling back the bowstring in Step 2 imparts kinetic energy to the arrow and causes it to leave the bow. This step thus continues to develop the nonlinear response of the system, but now with dynamic effects included.

In this case it is obvious that each nonlinear general analysis step must use the state at the end of the previous nonlinear general analysis step as its initial condition. For example, the dynamic part of the history has no loading—the dynamic response is caused by the release of some of the strain energy stored in the static steps. This effect introduces a natural order dependency in the model: nonlinear general analysis steps follow one another, in the order in which the events they define occur, with linear perturbation analysis steps inserted at the appropriate times in this sequence to investigate the linear behavior of the system at those times.

A more complex load history is illustrated in *Figure 2*, which shows a schematic representation of the steps in the manufacture and use of a stainless steel sink.

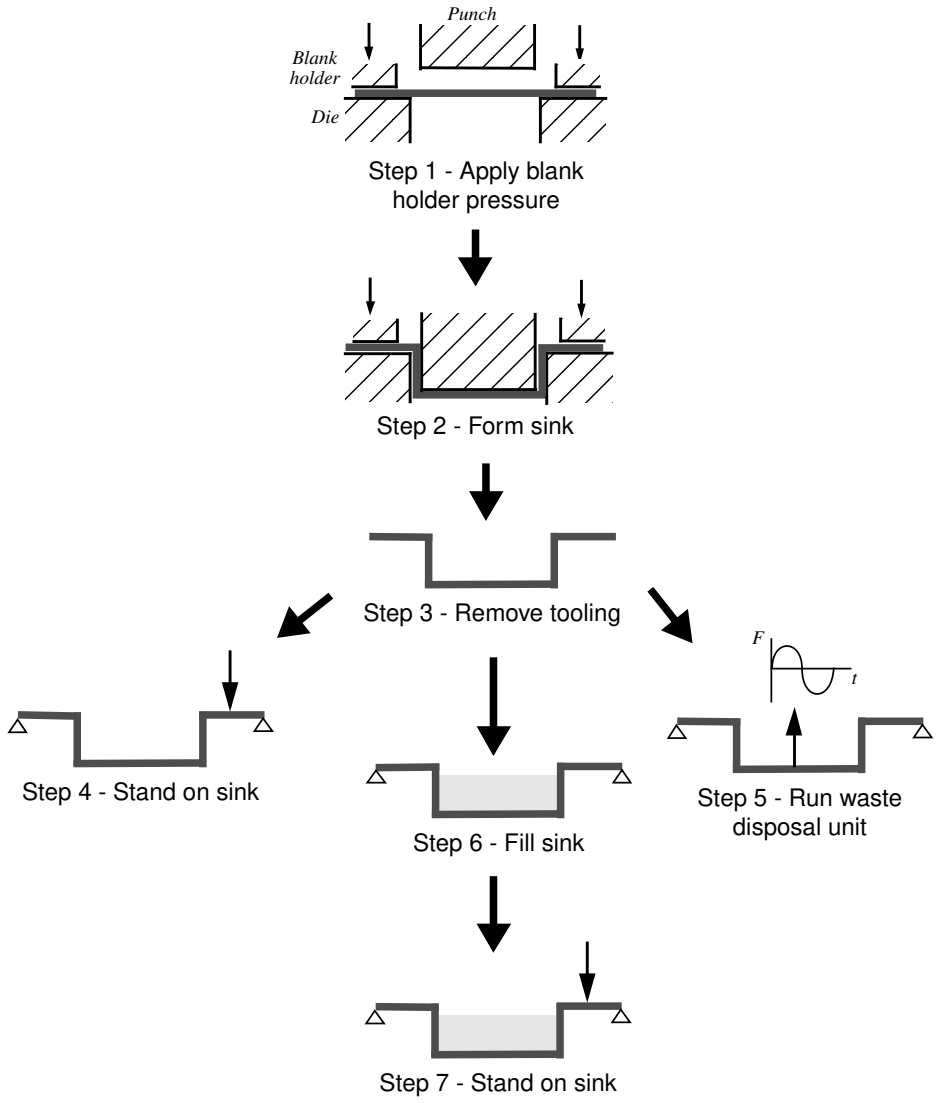


Figure 2: Steps in the manufacture and use of a sink.

The sink is formed from sheet steel using a punch, a die, and a blank holder. This forming simulation will consist of a number of general steps. Typically, the first step may involve the application of blank holder pressure, and the punching operation will be simulated in the second step. The third step will involve the removal of the tooling, allowing the sink to spring back into its final shape. Each of these steps is a general step since together they model a sequential load history, where the starting condition for each step is the ending condition from the previous step. These steps obviously include many nonlinear effects (plasticity, contact, large deformations). At the end of the third step, the sink will contain residual stresses and inelastic strains caused by the forming process. Its thickness will also vary as a direct result of the manufacturing process.

The sink is then installed: boundary conditions would be applied around the edge of the sink where it is attached to the worktop. The response of the sink to a number of different loading conditions may be of interest and has to be simulated. For example, a simulation may need to be performed to ensure that the sink does not break if someone stands on it. Step 4 would, therefore, be a linear perturbation step analyzing the static response of the sink to a local pressure load. Remember that the results from Step 4 will be perturbations from the state of the sink after the forming process; do not be surprised, for example, if the displacement of the center of the sink in this step is say only 2 mm, but you know that the sink deformed much more than that since the start of the forming simulation. This hypothetical

2 mm deflection is just the additional deformation from the sink's final configuration after the forming process (i.e., the end of Step 3) caused by the weight of the person. The total deflection, measured from the undeformed sheet's configuration, is the sum of this 2 mm and the deflection at the end of Step 3.

The sink may also be fitted with a waste disposal unit, so its steady-state dynamic response to a harmonic load at certain frequencies must be simulated. Step 5 would, therefore, be a second linear perturbation step using the direct steady-state dynamics procedure with a load applied at the point of attachment of the disposal unit. The base state for this step is the state at the end of the previous general step—that is, at the end of the forming process (Step 3). The response in the previous perturbation step (Step 4) is ignored. The two perturbation steps are, therefore, separate, independent simulations of the sink's response to loads applied to the base state of the model.

If another general step is included in the analysis, the condition of the structure at the start of the step is that at the end of the previous general step (Step 3). Step 6 could, therefore, be a general step with loads modeling the sink being filled with water. The response in this step may be linear or nonlinear. Following this general step, Step 7 could be a simulation repeating the analysis performed in Step 4. However, in this case the base state (the state of the structure at the end of the previous general step) is the state of the model at the end of Step 6. Therefore, the response will be that of a full sink, rather than an empty one. Performing another steady-state dynamics simulation would produce inaccurate results because the mass of the water, which would change the response considerably, would not be considered in the analysis.

The following procedures in Abaqus/Standard are always linear perturbation steps:

- linear eigenvalue buckling,
- frequency extraction,
- transient modal dynamics,
- random response,
- response spectrum, and
- steady-state dynamics.

The static procedure can be either a general or linear perturbation procedure.

Example: vibration of a piping system

This section examines the vibrational frequencies of a piping system.

In this section:

- *Vibrational frequencies analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Monitoring the job*
- *Postprocessing*

Vibrational frequencies analysis

In this example you will study the vibrational frequencies of a 5 m long section of a piping system.

The pipe is made of steel and has an outer diameter of 18 cm and a 2 cm wall thickness (see *Figure 1*).

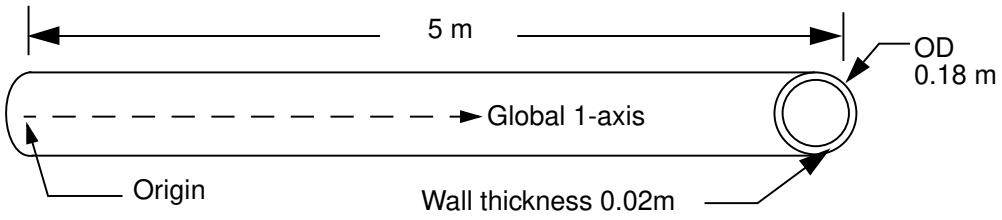


Figure 1: Portion of piping system being analyzed.

It is clamped firmly at one end and can move only axially at the other end. This 5 m portion of the piping system may be subjected to harmonic loading at frequencies up to 50 Hz. The lowest vibrational mode of the unloaded structure is 40.1 Hz, but this value does not consider how the loading applied to the piping structure may affect its response. To ensure that the section does not resonate, you have been asked to determine the magnitude of the in-service load that is required so that its lowest vibrational mode is higher than 50 Hz. You are told that the section of pipe will be subjected to axial tension when in service. Start by considering a load magnitude of 4 MN.

The lowest vibrational mode of the pipe will be a sine wave deformation in any direction transverse to the pipe axis because of the symmetry of the structure's cross-section. You will use three-dimensional beam elements to model the pipe section.

The analysis requires a natural frequency extraction. Thus, you will use Abaqus/Standard as your analysis product.

Preprocessing—creating the model with Abaqus/CAE

Create the model for this example using Abaqus/CAE.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Vibration of a piping system*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Vibration of a piping system**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Part geometry

Create a three-dimensional, deformable wire part. (Remember to use an approximate part size that is slightly larger than the largest dimension in your model.) Name the part `Pipe`, and use the **Create Lines: Connected** tool to sketch a horizontal line of length 5.0 m. Dimension the sketch as needed to ensure that the length is specified precisely.

Material and section properties

The pipe is made of steel with a Young's modulus of 200×10^9 Pa and a Poisson's ratio of 0.3. Create a linear elastic material named `Steel` with these properties. You must also define the density of the steel material (7800 kg/m^3) because eigenmodes and eigenfrequencies will be extracted in this simulation and a mass matrix is needed for this type of procedure.

Next, create a **Pipe** profile. Name the profile `PipeProfile`, choose **Thin-walled** as the formulation, and specify an outer radius of 0.09 m and a wall thickness of 0.02 m for the pipe.

Create a **Beam** section named `PipeSection`. In the **Edit Beam Section** dialog box, specify that section integration will be performed during the analysis and assign material `Steel` and profile `PipeProfile` to the section definition.

Finally, assign section `PipeSection` to the pipe. In addition, define the beam section orientation with the approximate $\mathbf{n_1}$-direction as the vector (0.0, 0.0, –1.0) (the default). In this model the actual $\mathbf{n_1}$-vector will coincide with this approximate vector.

Assembly and sets

Create a dependent instance of the part named `Pipe`. For convenience, create geometry sets that contain the vertices at the left and right ends of the pipe and name them `Left` and `Right`, respectively. These regions will be used later to assign loads and boundary conditions to the model.

Steps

In this simulation you need to investigate the eigenmodes and eigenfrequencies of the steel pipe section when a 4 MN tensile load is applied. Therefore, the analysis will be split into two steps:

Step 1. General step:	Apply a 4 MN tensile force.
Step 2. Linear perturbation step:	Calculate modes and frequencies.

Create a general static step named `Pull I` with the following step description: `Apply axial tensile load of 4.0 MN`. The actual magnitude of time in this step will have no effect on the results; unless the

model includes damping or rate-dependent material properties, “time” has no physical meaning in a static analysis procedure. Therefore, use a time period of 1.0. Include the effects of geometric nonlinearity, and specify an initial increment size that is 1/10 the total step time. This causes Abaqus/Standard to apply 10% of the load in the first increment. Accept the default number of allowable increments.

You need to calculate the eigenmodes and eigenfrequencies of the pipe in its loaded state. Thus, create a second analysis step using the linear perturbation frequency extraction procedure. Name the step `Frequency I`, and give it the following description: `Extract modes and frequencies`. Although only the first (lowest) eigenmode is of interest, extract the first eight eigenmodes for the model.

Output requests

The default output database output requests created by Abaqus/CAE for each step will suffice; you do not need to create any additional output database output requests.

To request output to the restart file, select **Output->Restart Requests** from the main menu bar of the Step module. For the step labeled `Pull I`, write data to the restart file every 10 increments. For the step labeled `Frequency 1`, write restart data every increment.

Loads and boundary conditions

In the first step create a load named `Force` that applies a 4×10^6 N tensile force to the right end of the pipe section such that it deforms in the positive axial (global 1) direction. Forces are applied, by default, in the global coordinate system.

The pipe section is clamped at its left end. It is also clamped at the other end; however, the axial force must be applied at this end, so only degrees of freedom 2 through 6 (U2, U3, UR1, UR2, and UR3) are constrained. Apply the appropriate boundary conditions to sets **Left** and **Right** in the first step.

In the second step you require the natural frequencies of the extended pipe section. This does not involve the application of any perturbation loads, and the fixed boundary conditions are carried over from the previous general step. Therefore, you do not need to specify any additional loads or boundary conditions in this step.

Mesh and job definition

Seed and mesh the pipe section with 30 uniformly spaced second-order, pipe elements (PIPE32).

Before continuing, rename the model to `Original`. This model will later form the basis of the model used in the example discussed in *Example: restarting the pipe vibration analysis*.

Create a job named `Pipe` with the following description: `Analysis of a 5 meter long pipe under tensile load`.

Save your model in a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors and investigate the source of any warning messages, taking corrective action as necessary.

Monitoring the job

Check the **Job Monitor** as the job is running. When the analysis completes, its contents will look similar to *Figure 1*.

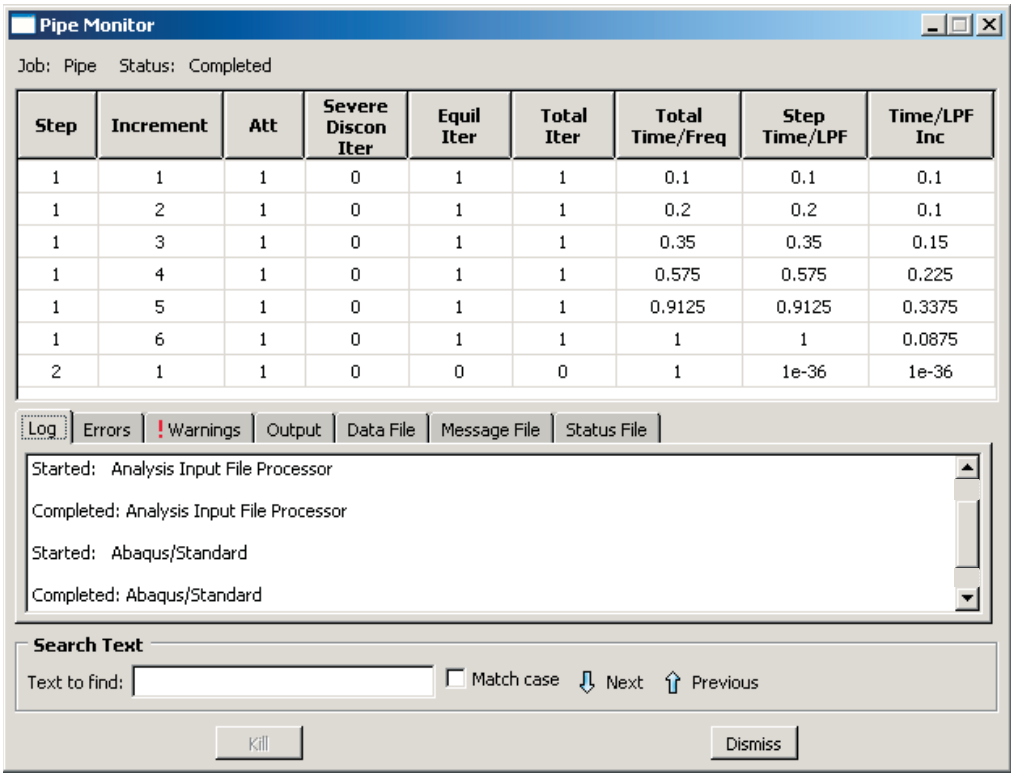


Figure 1: Job Monitor: original pipe vibration analysis.

Both steps are shown, and the time associated with the linear perturbation step (Step 2) is very small: the frequency extraction procedure, or any linear perturbation procedure, does not contribute to the general loading history of the model.

Postprocessing

Enter the Visualization module, and open the output database file, `Pipe.odb`, created by this job to review deformed shapes from the linear perturbation steps.

The Visualization module automatically uses the last available frame on the output database file. The results from the second step of this simulation are the natural mode shapes of the pipe and the corresponding natural frequencies. Plot the first mode shape.

1. From the main menu bar, select **Result->Step/Frame**.

   The **Step/Frame** dialog box appears.
2. Select step `Frequency I` and frame `Mode 1`.
3. Click **OK**.
4. From the main menu bar, select **Plot->Deformed Shape**.
5. Click the  tool in the toolbox to allow multiple plot states in the viewport, then click the  tool or select **Plot->Undeformed Shape** to add the undeformed shape plot to the existing deformed plot in the viewport.
6. Include node symbols on both plots (the superimpose options control the appearance of the undeformed shape when multiple plot states are displayed). Change the color of the node symbols to green and the symbol shape to a solid circle.
7. Click the auto-fit tool  so that the entire plot is rescaled to fit in the viewport.

The default view is isometric. Try rotating the model to find a better view of the first eigenmode, similar to that shown in *Figure 1*.

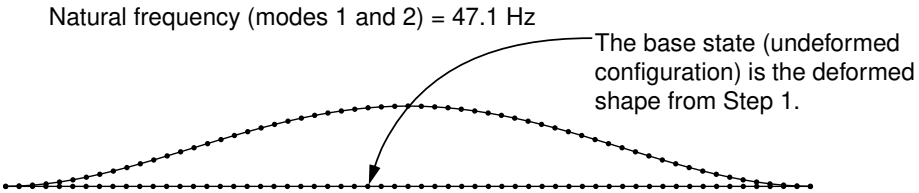


Figure 1: First and second eigenmode shapes of the pipe section under the tensile load (the modes lie in planes orthogonal to each other).

Since this is a linear perturbation step, the undeformed shape is the shape of the structure in the base state. This makes it easy to see the motion relative to the base state. Use the **Frame Selector** to plot the other mode shapes. You will discover that this model has many repeated eigenmodes. This is a result of the symmetric nature of the pipe's cross-section, which yields two eigenmodes for each natural frequency, corresponding to the 1–2 and 1–3 planes. The second eigenmode shape is shown in *Figure 1*. Some of the higher vibrational mode shapes are shown in *Figure 2*.

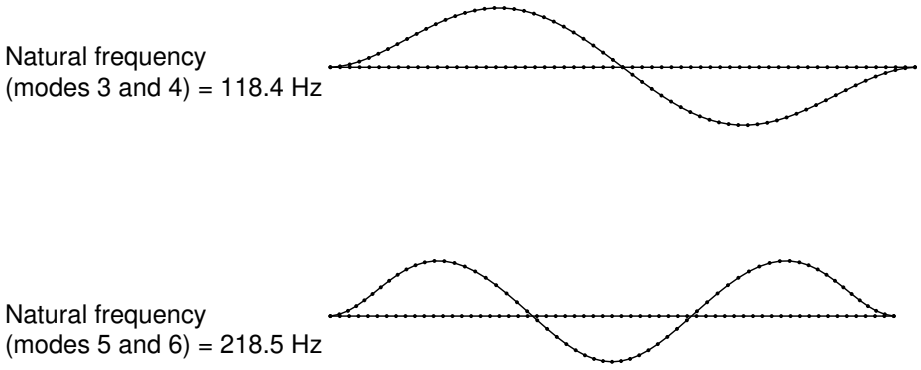


Figure 2: Shapes of eigenmodes 3 through 6; corresponding mode shapes lie in planes orthogonal to each other.

The natural frequency associated with each eigenmode is reported in the plot title. The lowest natural frequency of the pipe section when the 4 MN tensile load is applied is 47.1 Hz. The tensile loading has increased the stiffness of the pipe and, thus, increased the vibrational frequencies of the pipe section. This lowest natural frequency is within the frequency range of the harmonic loads; therefore, resonance of the pipe may be a problem when it is used with this loading.

You, therefore, need to continue the simulation and apply additional tensile load to the pipe section until you find the magnitude that raises the natural frequency of the pipe section to an acceptable level. Rather than repeating the analysis and increasing the applied axial load, you can use the restart capability in Abaqus to continue the load history of a prior simulation in a new analysis.

Restart analysis

Multistep simulations need not be defined in a single job. Indeed, it is usually desirable to run a complex simulation in stages. This allows you to examine the results and confirm that the analysis is performing as expected before continuing with the next stage. The Abaqus restart analysis capability allows a simulation to be restarted and the model's response to additional load history to be calculated.

The restart analysis capability is discussed in detail in *Restarting an Analysis*.

In this section:

- *The restart and state files*
- *Restarting an analysis*

The restart and state files

The Abaqus/Standard restart (`.res`) file and the Abaqus/Explicit state (`.abq`) file contain the information necessary to continue a previous analysis. In Abaqus/Explicit the package (`.pac`) file and the selected results (`.sel`) file are also used for restarting an analysis and must be saved upon completion of the first job. In addition, both products require the output database (`.odb`) file. Restart files can become very large for large models; when restart data are requested, they are written for every increment or interval by default. Thus, it is important to control the frequency at which restart data are written. Sometimes it is useful to allow data to be overwritten on the restart file during a step. This means that at the end of the analysis there is only one set of restart data for each step, corresponding to the state of the model at the end of each step. However, if the analysis is interrupted for some reason, such as a computer malfunction, the analysis can be continued from the point where restart data were last written.

Restarting an analysis

When restarting a simulation using the results of a previous analysis, you specify the particular point in the simulation's load history from which to restart the analysis.
The model used in the restart analysis, however, must be the same as the model used in the original analysis up to the restart location.

Specifically,

- the restart analysis model must not modify or add any geometry, mesh, materials, sections, beam section profiles, material orientations, beam section orientations, interaction properties, or constraints that are already defined in the original analysis model; and
- similarly, it must not modify any step, load, boundary condition, predefined field, or interaction at or before the restart location.

You may, however, define new sets and amplitude curves in the restart analysis model.

Continuing an interrupted run

The restart analysis continues directly from the specified step and increment of the previous analysis. If the given step and increment do not correspond to the end of the previous analysis (for example, if the analysis was interrupted by a computer malfunction), Abaqus will try to finish the original step before trying to simulate any new steps.

In Abaqus/Explicit 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), you can restart the run by using the **Recover** job type as shown in *Figure 1*.

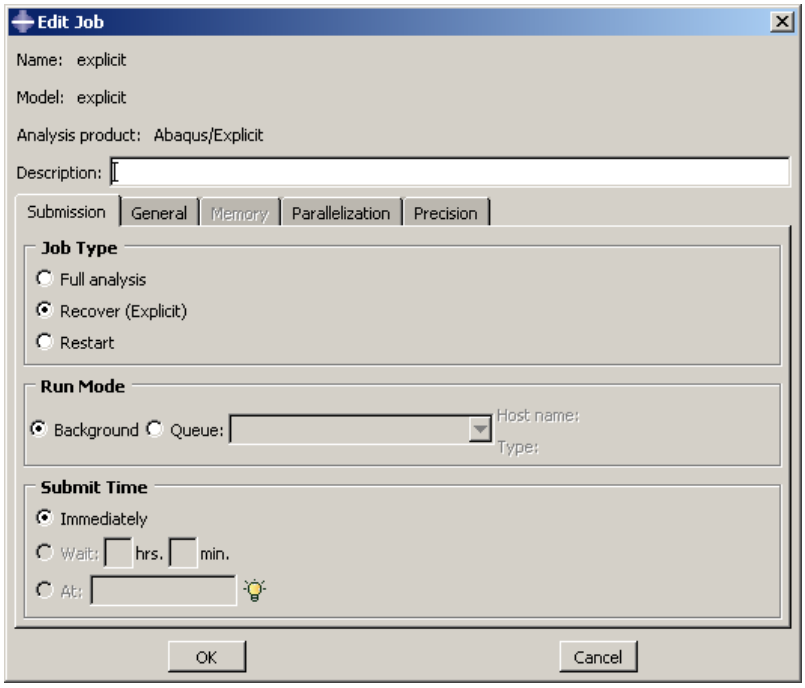


Figure 1: Recover job type.

Continuing with additional steps

If the previous analysis completed successfully and, having viewed the results, you want to add additional steps to the load history, the specified step and increment should be the last step and last increment of the previous analysis.

Changing an analysis

Sometimes, having viewed the results of the previous analysis, you may want to restart the analysis from an intermediate point and change the remaining load history in some manner—for example, to add more output requests, to change the loading, or to adjust the analysis controls. This can be necessary, for example, when a step has exceeded its maximum number of increments. If an analysis is restarted because the maximum number of increments has been exceeded, Abaqus/Standard thinks that the analysis is partway through a step, tries to complete the step, and promptly exceeds the maximum number of increments again.

In such situations you should indicate that the current step should be terminated at the specified step and increment. The simulation may then continue with the new steps. For example, if a step allowed only a maximum of 20 increments, which was less than the number necessary to complete the step, a new step should be defined in which the entire step definition, including applied loads and boundary conditions, is identical to that specified in the original run with the following exceptions:

- The number of increments should be increased.
- The total time of the new step should be the total time of the original step less the time completed in the first run. For example, if the time of the step as originally specified was 100 seconds and the analysis ran out of increments at a step time of 20 seconds, the duration of the step in the restart analysis should be 80 seconds.
- Any amplitude definitions specified in terms of step time need to be respecified to reflect the new time scale of the step. Amplitude definitions specified in terms of total time do not need to be changed, provided the modifications given above are used.

The magnitudes of any loads or prescribed boundary conditions remain unchanged since they are always total values in general analysis steps.

Example: restarting the pipe vibration analysis

This example demonstrates how to restart an analysis.
Take the pipe section example in *Example: vibration of a piping system* and restart the simulation, adding two additional steps of load history. The first simulation predicted that the piping section would be vulnerable to resonance when extended axially; you must now determine how much additional axial load will increase the pipe's lowest vibrational frequency to an acceptable level.

Step 3 will be a general step that increases the axial load on the pipe to 8 MN, and Step 4 will calculate the eigenmodes and eigenfrequencies again.

In this section:

- *Creating a restart analysis model*
- *Monitoring the job*
- *Postprocessing the restart analysis results*

Creating a restart analysis model

Open the model database file `Pipe.cae` (if it is not already open). Copy the model named `Original` to a model named `Restart`.
The modifications to this model are discussed next.

Model attributes

To perform a restart analysis, the model's attributes must be changed to indicate that the model should reuse data from a previous analysis. In the Model Tree, double-click the **Restart** model underneath the **Models** container. In the **Edit Model Attributes** dialog box that appears, specify that restart data will be read from the job `Pipe` and that the restart location will be the end of step `Frequency I`.

Step definitions

You will now create two new analysis steps. The first new step is a general static step; name the step `Pull II`, and insert it immediately after the step `Frequency I`. Give the step the following description: `Apply axial tensile load of 8.0 MN`; and set the time period for the step to `1.0` and the initial time increment to `0.1`.

The second new step is a frequency extraction step; name the step `Frequency II`, and insert it immediately after the step `Pull II`. Give the step the following description: `Extract modes and frequencies`; and use the Lanczos eigensolver to extract the first `8` natural modes and frequencies of the pipe.

Output requests

For the step `Pull II`, write data to the restart file every 10 increments.

Accept all other default output data requests.

Load definition

Modify the load definition so that the tensile load that is applied to the pipe is doubled in the second general static step (`Pull II`). To do this, expand the **Force** item underneath the **Loads** container in the Model Tree. In the list that appears, expand the **States** item. Double-click the step named **Pull II**, and change the value of the applied force to `8.0E+06` in this step. Open the **Load Manager**. The table is color coded according to the object's status. In particular, the cell corresponding to the load in step `Pull II` is colored yellow to distinguish it from the original load.

Job definition

Create a job named `PipeRestart` with the following description: `Restart analysis of a 5 meter long pipe under tensile load`. Set the job type to **Restart** if it is not already. (If the job type is not set to **Restart**, Abaqus/CAE ignores the model's restart attributes.)

Save your model in a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors and investigate the source of any warning messages, taking corrective action as necessary.

Monitoring the job

Again, check the **Job Monitor** as the job is running. When the analysis completes, its contents will look similar to *Figure 1*.

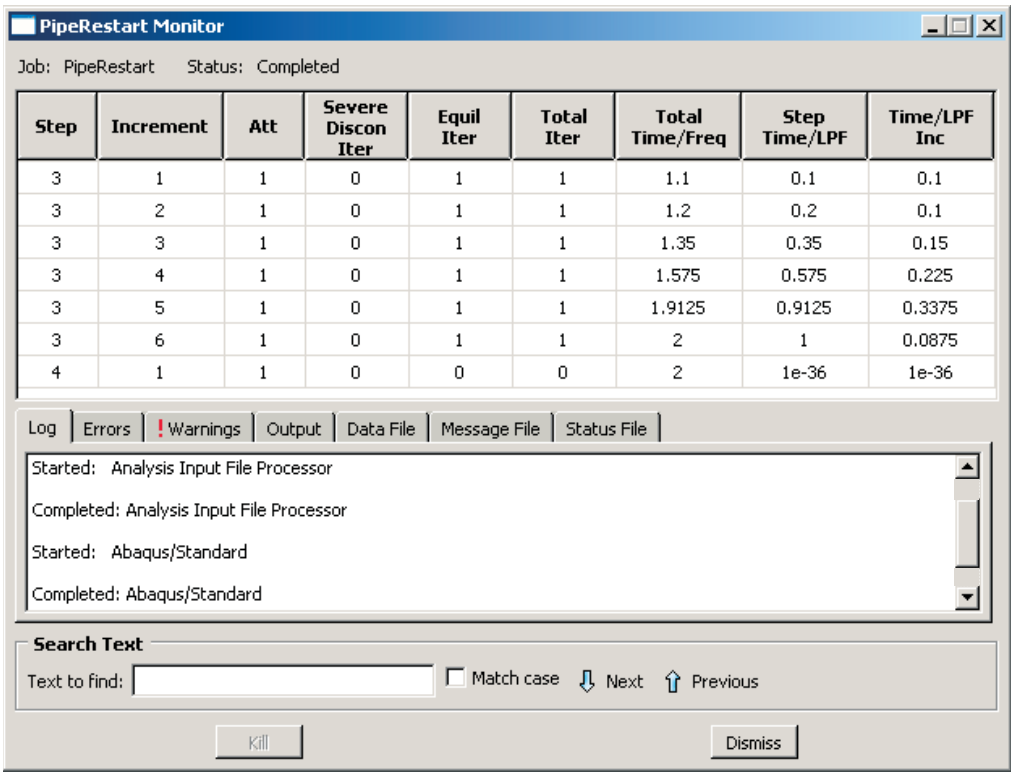


Figure 1: Job Monitor: restart pipe vibration analysis.

This analysis starts at Step 3 since Steps 1 and 2 were completed in the previous analysis. There are now two output database (`.odb`) files associated with this simulation. Data for Steps 1 and 2 are in the file `Pipe.odb`; data for Steps 3 and 4 are in the file `PipeRestart.odb`. When plotting results, you need to remember which results are stored in each file, and you need to ensure that Abaqus/CAE is using the correct output database file.

Postprocessing the restart analysis results

Switch to the Visualization module, and open the output database file from the restart analysis, `PipeRestart.odb`.

In this section:

- *Plotting the eigenmodes of the pipe*
- *Plotting X–Y graphs from field data for selected steps*

Plotting the eigenmodes of the pipe

Plot the same six eigenmode shapes of the pipe section for this simulation as were plotted in the previous analysis. The eigenmode shapes can be plotted using the procedures described for the original analysis. These eigenmodes and their natural frequencies are shown in *Figure 1*; again, the corresponding mode shapes lie in planes orthogonal to each other.

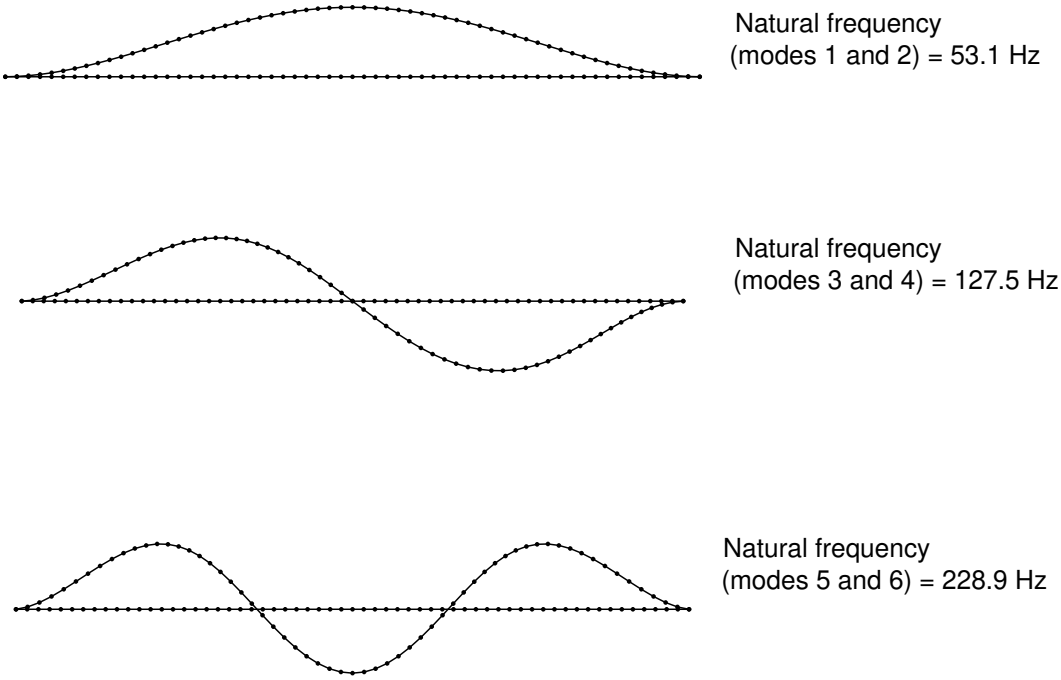


Figure 1: Shapes and frequencies of eigenmodes 1 through 6 with 8 MN tensile load.

Under 8 MN of axial load, the lowest mode is now at 53.1 Hz, which is greater than the required minimum of 50 Hz. If you want to find the exact load at which the lowest mode is just above 50 Hz, you can repeat this restart analysis and change the value of the applied load.

Plotting X–Y graphs from field data for selected steps

Use the field data stored in the output database files, `Pipe.odb` and `PipeRestart.odb`, to plot the history of the axial stress in the pipe for the whole simulation.

Generate a history plot of the axial stress in the pipe for the restart analysis

1. In the Results Tree, double-click **XYData**.

   The **Create XY Data** dialog box appears.
2. Select **ODB field output** from this dialog box, and click **Continue** to proceed.

   The **XY Data from ODB Field Output** dialog box appears.
3. In the **Variables** tabbed page of this dialog box, accept the default selection of `Integration Point` for the variable position and select **S11** from the list of available stress components.
4. At the bottom of the dialog box, toggle **Select** for the section point and click **Settings** to choose a section point.
5. In the **Field Report Section Point Settings** dialog box that appears, select the category `beam` and choose any of the available section points for the pipe cross-section. Click **OK** to exit this dialog box.
6. In the **Elements/Nodes** tabbed page of the **XY Data from ODB Field Output** dialog box, select **Element labels** as the selection **Method**. There are 30 elements in the model, and they are numbered consecutively from 1 to 30. Enter any element number (for example, `25`) in the **Element labels** text field that appears on the right side of the dialog box.
7. Click **Active Steps/Frames**, and select `Pull II` as the only step from which to extract data.
8. At the bottom of the **XY Data from ODB Field Output** dialog box, click **Plot** to see the history of axial stress in the element.

   The plot traces the axial stress history at each integration point of the element in the restart analysis. Since the restart is a continuation of an earlier job, it is often useful to view the results from the entire (original and restarted) analysis.

Generate a history plot of the axial stress in the pipe for the entire analysis

1. Save the current plot by clicking **Save** at the bottom of the **XY Data from ODB Field Output** dialog box. Two curves are saved (one for each integration point), and default names are given to the curves.
2. Rename either curve `RESTART`, and delete the other curve.
3. From the main menu bar, select **File->Open**; or use the  tool in the File toolbar to open the file `Pipe.odb`.
4. Following the procedure outlined above, save the plot of the axial stress history for the same element and integration/section point used above. Name this plot `ORIGINAL`.
5. In the Results Tree, expand the **XYData** container.

   The `ORIGINAL` and `RESTART` curves are listed underneath.
6. Select both plots with **[Ctrl] + Click**. Click mouse button 3, and select **Plot** from the menu that appears to create a plot of axial stress history in the pipe for the entire simulation.
7. To change the style of the line, open the **Curve Options** dialog box.
8. For the `RESTART` curve, select a dotted line style.
9. Click **Dismiss** to close the dialog box.
10. To change the axis titles, open the **Axis Options** dialog box.

    In this dialog box, switch to the **Title** tabbed page.
11. Change the *X*-axis title to `TOTAL TIME`, and change the *Y*-axis title to `STRESS S11`.

12. Click **Dismiss** to close the dialog box.

The plot created by these commands is shown in *Figure 1*. The axial stress history of the same element during Step 3 can be plotted by itself by selecting only the `RESTART` curve (see *Figure 2*).

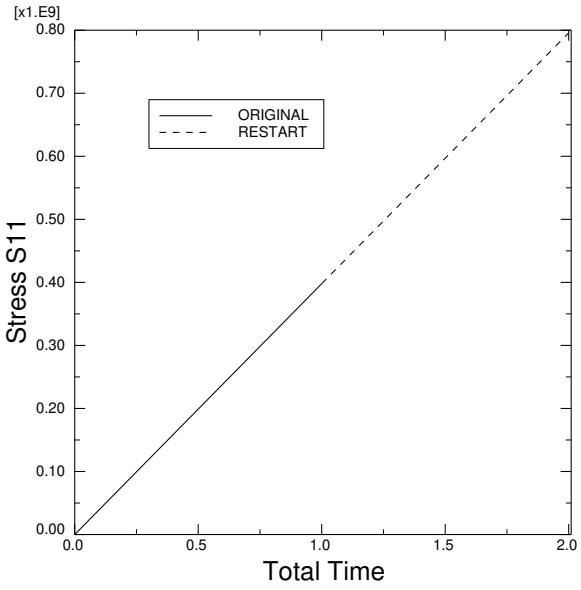


Figure 1: History of axial stress in the pipe.

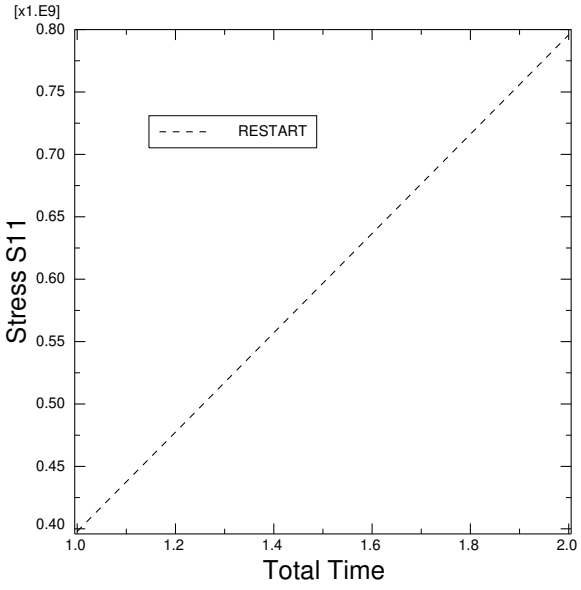


Figure 2: History of axial stress in the pipe during Step 3.

Related Abaqus examples

Related Abaqus examples are available.

- *Deep drawing of a cylindrical cup*
- *Linear analysis of the Indian Point reactor feedwater line*
- *Vibration of a cable under tension*
- *Random response to jet noise excitation*

Summary

This section presents a summary of the concepts.

- An Abaqus simulation can include any number of steps.
- Implicit and explicit steps are not allowed in the same analysis job.
- An analysis step is a period of “time” during which the response of the model to a specified set of loads and boundary conditions is calculated. The character of this response is determined by the particular analysis procedure used during the step.
- The response of a structure in a general analysis step may be either linear or nonlinear.
- The starting condition for each general step is the ending condition of the previous general step. Thus, the model's response evolves during a sequence of general steps in a simulation.
- Linear perturbation steps (available only in Abaqus/Standard) calculate the linear response of the structure to a perturbation load. The response is reported relative to the base state defined by the condition of the model at the end of the last general step.
- Analyses can be restarted as long as a restart file is saved. Restart files can be used to continue an interrupted analysis or to add additional load history to the simulation.

Contact

Many engineering problems involve contact between two or more components.
In these problems a force normal to the contacting surfaces acts on the two bodies when they touch each other. If there is friction between the surfaces, shear forces may be created that resist the tangential motion (sliding) of the bodies. The general aim of contact simulations is to identify the areas on the surfaces that are in contact and to calculate the contact pressures generated.

In a finite element analysis contact conditions are a special class of discontinuous constraint, allowing forces to be transmitted from one part of the model to another. The constraint is discontinuous because it is applied only when the two surfaces are in contact. When the two surfaces separate, no constraint is applied. The analysis has to be able to detect when two surfaces are in contact and apply the contact constraints accordingly. Similarly, the analysis must be able to detect when two surfaces separate and remove the contact constraints.

In this section:

- *Contact capabilities in Abaqus*
- *Defining surfaces*
- *Interaction between surfaces*
- *Defining contact in Abaqus/Standard*
- *Modeling issues for rigid surfaces in Abaqus/Standard*
- *Abaqus/Standard 2D example: forming a channel*
- *General contact in Abaqus/Standard*
- *Abaqus/Standard 3D example: shearing of a lap joint*
- *Defining contact in Abaqus/Explicit*
- *Modeling considerations in Abaqus/Explicit*
- *Abaqus/Explicit example: circuit board drop test*
- *Compatibility between Abaqus/Standard and Abaqus/Explicit*
- *Related Abaqus examples*
- *Suggested reading*
- *Summary*

Contact capabilities in Abaqus

Contact simulations in Abaqus/Standard can either be surface based or contact element based. Contact simulations in Abaqus/Explicit are surface based only. In this guide, surface-based contact is discussed.

Surface-based contact can utilize either the general (“automatic”) contact algorithm or the contact pair algorithm. The general contact algorithm allows for a highly automated contact definition, where contact is based on an automatically generated all-inclusive surface definition. Conversely, the contact pair algorithm requires you to explicitly pair surfaces that may potentially come into contact. Both algorithms require specification of contact properties between surfaces (for example, friction).

The discussion of contact in this guide addresses the contact pair approach and general contact in Abaqus/Standard and general contact in Abaqus/Explicit.

Defining surfaces

Surfaces are created from the element faces of the underlying material. The discussion that follows assumes that the surfaces will be defined in Abaqus/CAE. Restrictions on the types of surfaces that can be created in Abaqus are discussed in *Surface Definition*; please read them before beginning a contact simulation.

Surfaces on continuum elements

For two- and three-dimensional solid continuum elements you specify which regions of a part form the contact surface by selecting the regions of a part instance in the viewport.

Surfaces on structural, surface, and rigid elements

There are four ways to define surfaces on structural, surface, and rigid elements: using single-sided surfaces, double-sided surfaces, edge-based surfaces, and node-based surfaces.

Using single-sided surfaces, you specify which side of the element forms the contact surface. The side in the direction of the positive element normal is called SPOS, while the side in the direction of the negative element normal is called SNEG, as shown in *Figure 1*. The connectivity of an element defines the positive element normal, as discussed in *Using Shell Elements*. The positive element normals can be viewed in Abaqus/CAE.

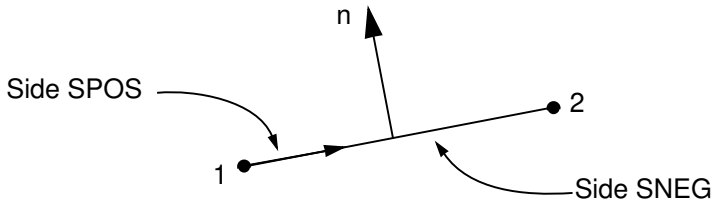


Figure 1: Surfaces on a two-dimensional shell or rigid element.

Double-sided contact surfaces are more general because both the SPOS and SNEG faces and all free edges are included automatically as part of the contact surface. Contact can occur on either face or on the edges of the elements forming the double-sided surface. For example, a secondary node can start on one side of a double-sided surface and then travel around the perimeter to the other side during the course of an analysis. Currently, double-sided surfaces can be defined only on three-dimensional shell, membrane, surface, and rigid elements. In Abaqus/Explicit the general contact algorithm and self-contact in the contact pair algorithm enforce contact on both sides of all shell, membrane, surface, and rigid surface facets, even if they are defined as single-sided. Double-sided contact surfaces cannot be used with the default contact formulation in Abaqus/Standard, but they can be used with certain optional contact formulations; see *About Contact Pairs in Abaqus/Standard* for more information.

Edge-based surfaces consider contact on the perimeter edges of the model. They can be used to model contact on a shell edge, for example. Alternatively, node-based surfaces, which define contact between a set of nodes and a surface, can be used to achieve the same effect, as shown in *Figure 2*.

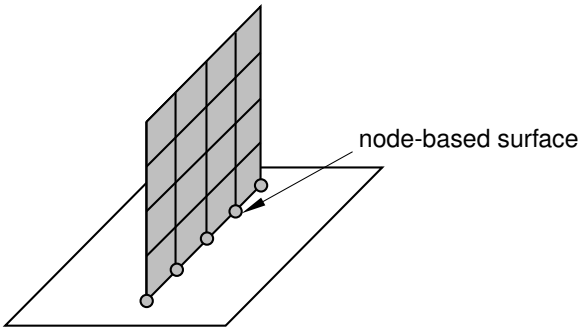


Figure 2: Node-based region for contact on a shell edge.

Rigid surfaces

Rigid surfaces are the surfaces of rigid bodies. They can be defined as an analytical shape, or they can be based on the underlying surfaces of elements associated with the rigid body.

Analytical rigid surfaces have three basic forms. In two dimensions the specific form of an analytical rigid surface is a two-dimensional, segmented rigid surface. The cross-section of the surface is defined in the two-dimensional plane of the model using straight lines, circular arcs, and parabolic arcs. The cross-section of a three-dimensional rigid surface is defined in a user-specified plane in the same manner used for two-dimensional surfaces. Then, this cross-section is swept around an axis to form a surface of revolution or extruded along a vector to form a long three-dimensional surface as shown in *Figure 3*.

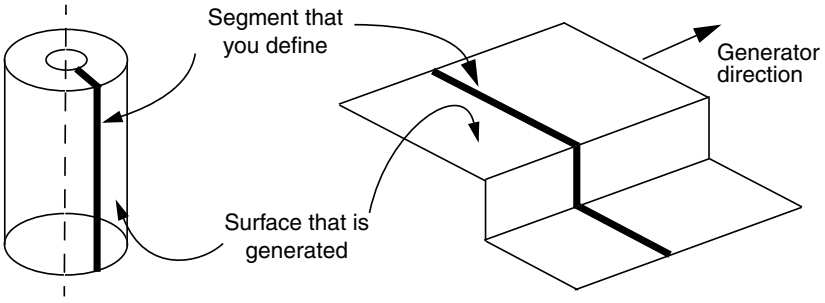


Figure 3: Analytical rigid surfaces.

The benefit of analytical rigid surfaces is that they are defined by only a small number of geometric points and are computationally efficient. However, in three dimensions the range of shapes that can be created with them is limited.

Discretized rigid surfaces are based on the underlying elements that make up a rigid body; thus, they can be more geometrically complex than analytical rigid surfaces. Discretized rigid surfaces are defined in exactly the same manner as surfaces on deformable bodies.

Interaction between surfaces

The interaction between contacting surfaces consists of two components: one normal to the surfaces and one tangential to the surfaces.
The tangential component consists of the relative motion (sliding) of the surfaces and, possibly, frictional shear stresses. Each contact interaction can refer to a contact property that specifies a model for the interaction between the contacting surfaces. There are several contact interaction models available in Abaqus; the default model is frictionless contact with no bonding.

In this section:

- *Behavior normal to the surfaces*
- *Sliding of the surfaces*
- *Friction models*
- *Other contact interaction options*
- *Surface-based constraints*

Behavior normal to the surfaces

The distance separating two surfaces is called the clearance. The contact constraint is applied in Abaqus when the clearance between two surfaces becomes zero. The contact formulation imposes no limit on the magnitude of contact pressure that can be transmitted between the surfaces. The surfaces separate when the contact pressure between them becomes zero or negative, and the constraint is removed. This behavior, referred to as "hard" contact, is the default contact behavior in Abaqus and is summarized in the contact pressure-clearance relationship shown in *Figure 1*.

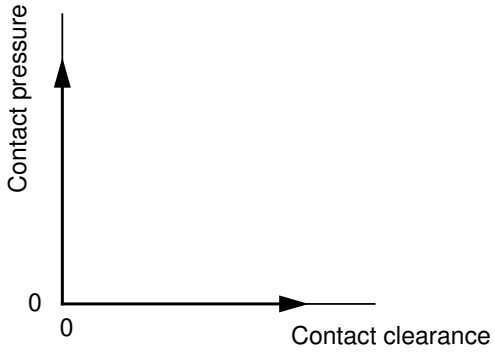


Figure 1: Contact pressure-clearance relationship for "hard" contact.

By default, "hard" contact is directly enforced when using contact pairs in Abaqus/Standard. The dramatic change in contact pressure that occurs when a contact condition changes from "open" (a positive clearance) to "closed" (clearance equal to zero) sometimes makes it difficult to complete contact simulations in Abaqus/Standard; the same is not true for Abaqus/Explicit since iteration is not required for explicit methods. Alternative enforcement methods (e.g., penalty) are available for contact pairs, as discussed in *Contact Constraint Enforcement Methods in Abaqus/Standard*. Penalty enforcement of the contact constraints is the only option available for general contact. Other sources of information include *Common Difficulties Associated with Contact Modeling in Abaqus/Standard*, *Common Difficulties Associated with Contact Modeling Using Contact Pairs in Abaqus/Explicit*, the "Modeling Contact with Abaqus/Standard" lecture notes, and the "Advanced Topics: Abaqus/Explicit" lecture notes.

Sliding of the surfaces

In addition to determining whether contact has occurred at a particular point, an Abaqus analysis also must calculate the relative sliding of the two surfaces.

This can be a very complex calculation; therefore, Abaqus makes a distinction between analyses where the magnitude of sliding is small and those where the magnitude of sliding may be finite. It is much less expensive computationally to model problems where the sliding between the surfaces is small. What constitutes “small sliding” is often difficult to define, but a general guideline to follow is that problems can use the “small-sliding” approximation if a point contacting a surface does not slide more than a small fraction of a typical element dimension.

Friction models

When surfaces are in contact, they usually transmit shear as well as normal forces across their interface. Thus, the analysis may need to take frictional forces, which resist the relative sliding of the surfaces, into account. *Coulomb friction* is a common friction model used to describe the interaction of contacting surfaces. The model characterizes the frictional behavior between the surfaces using a coefficient of friction, μ.

The default friction coefficient is zero. The tangential motion is zero until the surface traction reaches a critical shear stress value, which depends on the normal contact pressure, according to the following equation:

$$\tau_{\text{crit}} = \mu p,$$

where μ is the coefficient of friction and p is the contact pressure between the two surfaces. This equation gives the limiting frictional shear stress for the contacting surfaces. The contacting surfaces will not slip (slide relative to each other) until the shear stress across their interface equals the limiting frictional shear stress, μp. For most surfaces μ is normally less than unity. Coulomb friction can be defined with μ or τ_{crit}. The solid line in *Figure 1* summarizes the behavior of the Coulomb friction model: there is zero relative motion (slip) of the surfaces when they are sticking (the shear stresses are below μp). Optionally, a friction stress limit can be specified if both contacting surfaces are element-based surfaces.

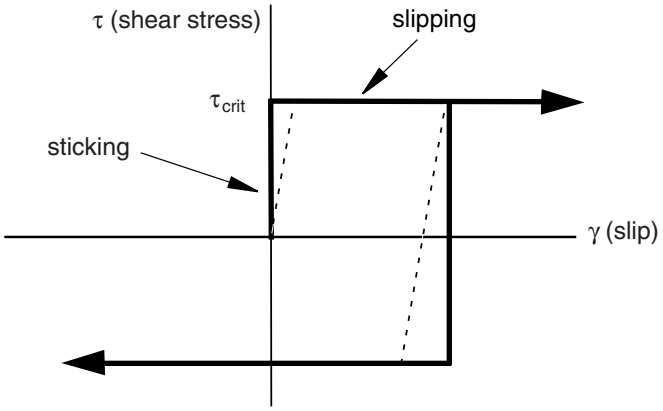


Figure 1: Frictional behavior.

In Abaqus/Standard the discontinuity between the two states—sticking or slipping—can result in convergence problems during the simulation. You should include friction in your Abaqus/Standard simulations only when it has a significant influence on the response of the model. If your contact simulation with friction encounters convergence problems, one of the first modifications you should try in diagnosing the difficulty is to rerun the analysis without friction. In general, friction presents no additional computational difficulties for Abaqus/Explicit.

Simulating ideal friction behavior can be very difficult; therefore, by default in most cases, Abaqus uses a penalty friction formulation with an allowable "elastic slip," shown by the dotted line in *Figure 1*. The "elastic slip" is the small amount of relative motion between the surfaces that occurs when the surfaces should be sticking. Abaqus automatically chooses the penalty stiffness (the slope of the dotted line) so that this allowable "elastic slip" is a very small fraction of the characteristic element length. The penalty friction formulation works well for most problems, including most metal forming applications.

In those problems where the ideal stick-slip frictional behavior must be included, the "Lagrange" friction formulation can be used in Abaqus/Standard and the kinematic friction formulation can be used in Abaqus/Explicit. The "Lagrange" friction formulation is more expensive in terms of the computer resources used because Abaqus/Standard uses additional variables for each surface node with frictional contact. In addition, the solution converges more slowly so that additional iterations are usually required. This friction formulation is not discussed in this guide.

Often the friction coefficient at the initiation of slipping from a sticking condition is different from the friction coefficient during established sliding. The former is typically referred to as the static friction coefficient, and the latter is referred to as the kinetic friction coefficient. In Abaqus an exponential decay law is available to model the transition between static and kinetic friction (see *Figure 2*). This friction formulation is not discussed in this guide.

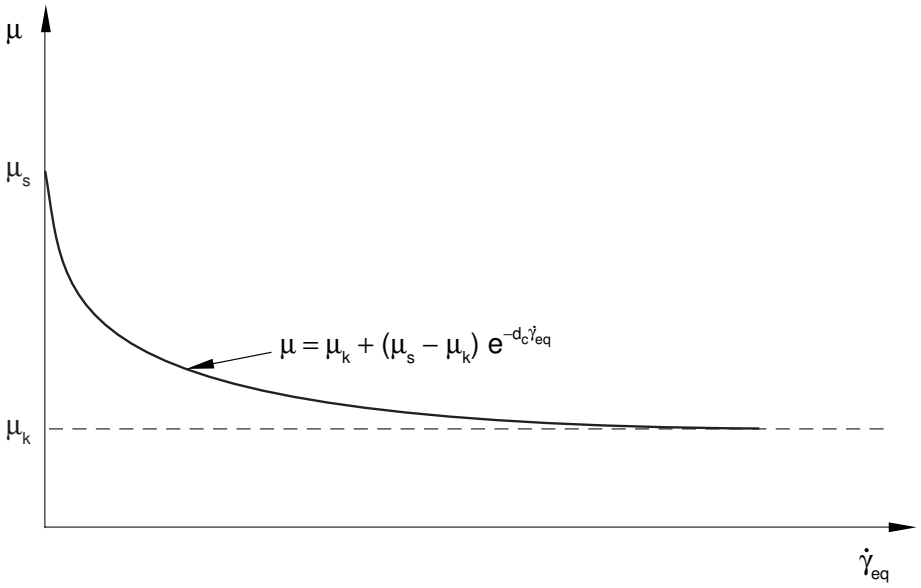


Figure 2: Exponential decay friction model.

In Abaqus/Standard the inclusion of friction in a model adds unsymmetric terms to the system of equations being solved. If μ is less than about 0.2, the magnitude and influence of these terms are quite small and the regular, symmetric solver works well (unless the contact surface has high curvature). For higher coefficients of friction, the unsymmetric solver is invoked automatically because it will improve the convergence rate. The unsymmetric solver requires twice as much computer memory and scratch disk space as the symmetric solver. Large values of μ generally do not cause any difficulties in Abaqus/Explicit.

Other contact interaction options

The other contact interaction models available in Abaqus depend on the analysis product and the algorithm used and may include adhesive contact behavior, softened contact behavior, fasteners (for example, spot welds), and viscous contact damping. These options are not discussed in this guide. Details about them can be found in the *Abaqus Interactions Guide*.

Surface-based constraints

Tie constraints are used to tie together two surfaces for the duration of a simulation. Each node on the secondary surface is constrained to have the same motion as the point on the main surface to which it is closest. For a structural analysis this means the translational (and, optionally, the rotational) degrees of freedom are constrained.

Abaqus uses the undeformed configuration of the model to determine which secondary nodes are tied to the main surface. By default, all secondary nodes that lie within a given distance of the main surface are tied. The default distance is based on the typical element size of the main surface. This default can be overridden in one of two ways: by specifying the distance within which secondary nodes must lie from the main surface to be constrained or by specifying the name of a set containing the nodes that will be constrained.

Secondary nodes can also be adjusted so that they lie exactly on the main surface. If secondary nodes have to be adjusted by distances that are a large fraction of the length of the side of the element (to which the secondary node is attached), the element can become severely distorted; avoid large adjustments if possible.

Tie constraints are particularly useful for rapid mesh refinement between dissimilar meshes.

Defining contact in Abaqus/Standard

The first step in defining contact pairs (and optionally general contact) in Abaqus/Standard is to define the surfaces of the bodies that could potentially come into contact.
The next step is to specify the surfaces that interact with each other; these are the contact interactions. The final step is to define the mechanical property models that govern the behavior of the surfaces when they are in contact.

The definition of surfaces is optional for general contact since an all-inclusive element-based surface is automatically created when general contact is used. You can use specific surface pairings to include regions not included in the default surface, to preclude interactions between different regions of a model, or to override global contact property assignments. For example, if you want to apply a certain friction coefficient to all but a few surfaces in your model, you can assign the primary friction coefficient globally and then override this property for a given pair of user-defined surfaces.

In this section:

- *Contact interactions*
- *Secondary and main surfaces*
- *Contact discretization*
- *Small and finite sliding*
- *Element selection*
- *Contact algorithm*

Contact interactions

In an Abaqus/Standard simulation possible contact is defined by assigning the surface names to a contact interaction (contact pair approach) or by invoking an automatically-defined all-inclusive element-based surface as the contact domain (general contact approach). For both contact approaches, each contact interaction must refer to a contact property, in much the same way that each element must refer to an element property. Constitutive behavior, such as the contact pressure-clearance relationship and friction, can be included in the contact property.

When you define a contact interaction, you must decide whether the magnitude of the relative sliding will be small or finite. The default is the more general finite-sliding formulation. The small-sliding formulation (available only for contact pairs) is appropriate if the relative motion of the two surfaces is less than a small proportion of the characteristic length of an element face. Using the small-sliding formulation when applicable results in a more efficient analysis.

Secondary and main surfaces

By default, contact pairs in Abaqus/Standard use a pure main-secondary contact algorithm: nodes on one surface (the secondary surface) cannot penetrate the segments that make up the other surface (the main surface), as shown in *Figure 1*. The algorithm places no restrictions on the main surface; it can penetrate the secondary surface between secondary nodes, as shown in *Figure 1*.

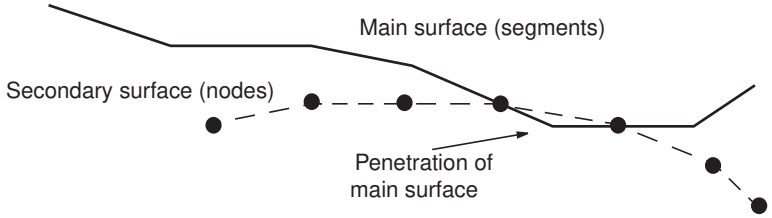


Figure 1: The main surface can penetrate the secondary surface.

Due to the strict main-secondary formulation, you must be careful to select the secondary and main surfaces correctly in order to achieve the best possible contact simulation. Some simple rules to follow are:

- the secondary surface should be the more finely meshed surface; and
- if the mesh densities are similar, the secondary surface should be the surface with the softer underlying material.

The general contact algorithm in Abaqus/Standard enforces contact in an average sense between interacting surfaces; Abaqus/Standard automatically assigns main and secondary roles.

Contact discretization

Abaqus/Standard offers two contact discretization methods: a traditional node-to-surface method and a surface-to-surface method. The node-to-surface discretization method defines contact conditions between each secondary node and the main surface. The surface-to-surface discretization method considers the shape of both the main and secondary surfaces when defining the contact constraints. The contact pair algorithm can use either discretization method; general contact uses only the surface-to-surface approach.

Small and finite sliding

When using the small-sliding formulation, Abaqus/Standard establishes the relationship between the secondary nodes and the main surface at the beginning of the simulation.
Abaqus/Standard determines which segment on the main surface will interact with each node on the secondary surface. It maintains these relationships throughout the analysis, never changing which main surface segments interact with which secondary nodes. If geometric nonlinearity is included in the model, the small-sliding algorithm accounts for any rotation and deformation of the main surface and updates the load path through which the contact forces are transmitted. If geometric nonlinearity is not included in the model, any rotation or deformation of the main surface is ignored and the load path remains fixed.

The finite-sliding contact formulation requires that Abaqus/Standard continually track which part of the main surface is in contact with each secondary node. This is a very complex calculation, especially if both the contacting bodies are deformable. The structures in such simulations can be either two- or three-dimensional. Abaqus/Standard can also model the finite-sliding self-contact of a deformable body. Such a situation occurs when a structure folds over onto itself.

The finite-sliding formulation for contact between a deformable body and a rigid surface is not as complex as the finite-sliding formulation for two deformable bodies. Finite-sliding simulations where the main surface is rigid can be performed for both two- and three-dimensional models.

Both the contact pair algorithm and general contact can consider either small or finite sliding effects.

Element selection

Selection of elements for contact depends heavily on the contact enforcement used. For example, for traditional contact formulations (i.e., the node-to-surface discretization) it is generally better to use first-order elements for those parts of a model that will form a secondary surface. Second-order elements can sometimes cause problems in this case because of the way these elements calculate consistent nodal loads for a constant pressure. The consistent nodal loads for a constant pressure, P, on a second-order, two-dimensional element with area A are shown in *Figure 1*.

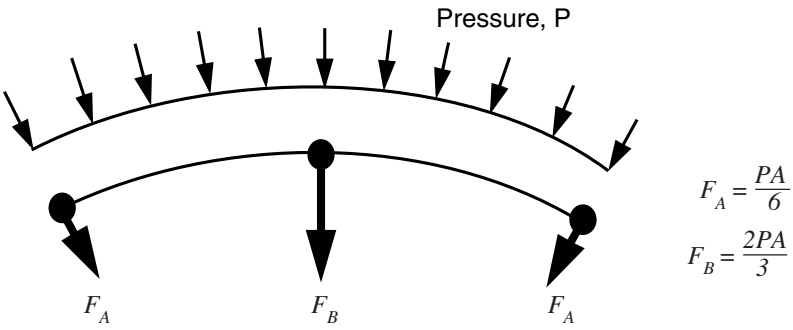


Figure 1: Equivalent nodal loads for a constant pressure on a two-dimensional, second-order element.

The node-to-surface contact formulation bases important decisions on the forces acting on the secondary nodes. It is difficult for the algorithm to tell if the force distribution shown in *Figure 1* represents a constant contact pressure or an actual variation across the element. The equivalent nodal forces for a three-dimensional, second-order brick element are even more confusing because they do not even have the same sign for a constant pressure, making it very difficult for the algorithm to work correctly, especially for nonuniform contact. Therefore, to avoid such problems, Abaqus/Standard automatically adds a midface node to any face of a second-order, three-dimensional brick or wedge element that defines a secondary surface when it is used with the node-to-surface formulation. The equivalent nodal forces for a second-order element face with a midface node have the same sign for a constant pressure, although they still differ considerably in magnitude.

The equivalent nodal forces for applied pressures on first-order elements always have a consistent sign and magnitude; therefore, there is no ambiguity about the contact state that a given distribution of nodal forces represents.

If you are using the node-to-surface formulation and your geometry is complicated and requires the use of an automatic mesh generator, the modified second-order tetrahedral elements (C3D10M) in Abaqus/Standard should be used. These elements are designed to be used in complex contact simulations; regular second-order tetrahedral elements (C3D10) have zero contact force at their corner nodes, leading to poor predictions of the contact pressures. The modified second-order tetrahedral elements can calculate the contact pressures accurately.

Regular second-order elements can generally be used without difficulty with the surface-to-surface formulation.

Contact algorithm

Understanding the algorithm Abaqus/Standard uses to solve contact problems will help you understand the diagnostic output in the message file and carry out contact simulations successfully.

The contact algorithm in Abaqus/Standard, which is shown in *Figure 1*, is built around the Newton-Raphson technique discussed in *Nonlinearity*.

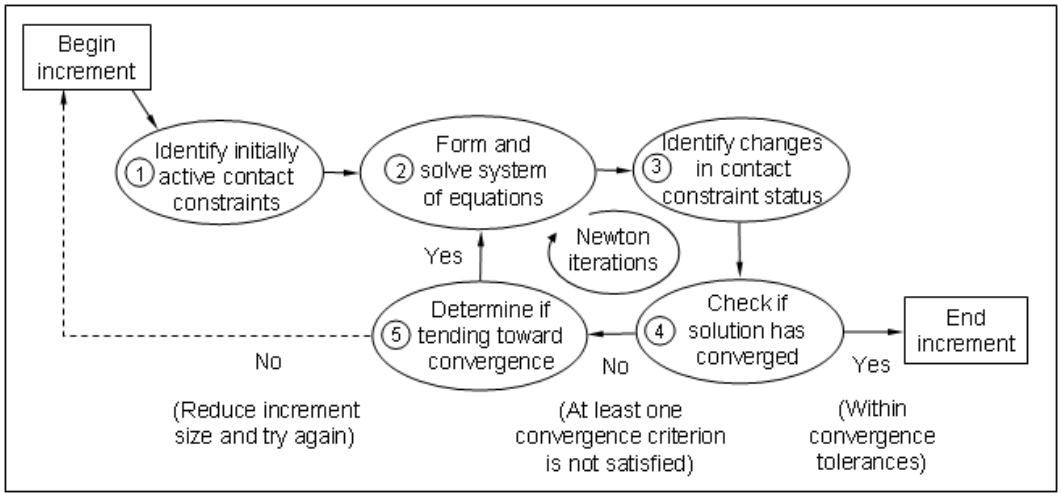


Figure 1: Contact algorithm in Abaqus/Standard.

Abaqus/Standard examines the state of all contact interactions at the start of each increment to establish whether secondary nodes are open or closed. If a node is closed, Abaqus/Standard determines whether it is sliding or sticking. Abaqus/Standard applies a constraint for each closed node and removes constraints from any node where the contact state changes from closed to open. Abaqus/Standard then carries out an iteration and updates the configuration of the model using the calculated corrections.

In the updated configuration Abaqus/Standard checks for changes in the contact conditions at the secondary nodes. Any node where the clearance after the iteration becomes negative or zero has changed status from open to closed. Any node where the contact pressure becomes negative has changed status from closed to open. If any contact changes are detected in the current iteration, Abaqus/Standard labels it a *severe discontinuity iteration*.

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 will continue to iterate until no severe discontinuities occur before checking for equilibrium.

The summary for each completed increment in the message and status files shows how many iterations were severe discontinuity iterations and how many were equilibrium iterations (an equilibrium iteration is one in which no severe discontinuities occur). The total number of iterations for an increment is the sum of these two. For some increments, you may find that all iterations are labeled severe discontinuity iterations (this occurs when small contact changes are detected in each iteration and equilibrium is ultimately satisfied).

Abaqus/Standard 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. The default limit for the maximum number of severe discontinuity iterations is 50, which in practice should always be more than the actual number of iterations in an increment.

Modeling issues for rigid surfaces in Abaqus/Standard

There are a number of issues that you should consider when modeling contact problems in Abaqus/Standard that involve rigid surfaces. These issues are discussed in detail in *Common Difficulties Associated with Contact Modeling in Abaqus/Standard*, but some of the more important issues are described here.

- The rigid surface is always the main surface in a contact interaction.
- The rigid surface should be large enough to ensure that secondary nodes do not slide off and "fall behind" the surface. If this happens, the solution usually will fail to converge. Extending the rigid surface or including corners along the perimeter (see *Figure 1*) will prevent secondary nodes from falling behind the main surface.

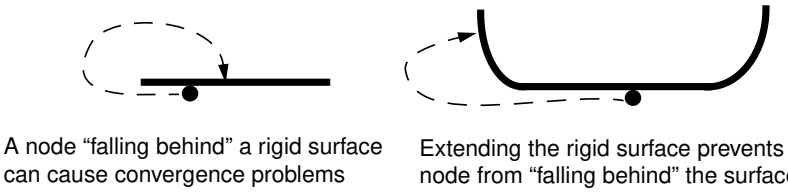


Figure 1: Extending rigid surfaces to prevent convergence problems.

- The deformable mesh must be refined enough to interact with any feature on the rigid surface. There is no point in having a 10 mm wide feature on the rigid surface if the deformable elements that will contact it are 20 mm across: the rigid feature will just penetrate into the deformable surface as shown in *Figure 2*.

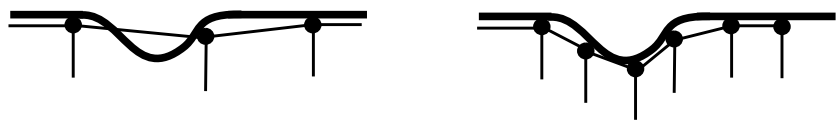


Figure 2: Modeling small features on the rigid surface.

With a sufficiently refined mesh on the deformable surface, Abaqus/Standard will prevent the rigid surface from penetrating the secondary surface.

- The contact algorithm in Abaqus/Standard requires the main surface of a contact interaction to be smooth. Rigid surfaces are always the main surface and so should always be smoothed. Abaqus/Standard does not smooth discrete rigid surfaces. The level of refinement controls the smoothness of a discrete rigid surface. Analytical rigid surfaces can be smoothed by defining a fillet radius that is used to smooth any sharp corners in the rigid surface definition (see *Figure 3*.)



Sharp corners in a rigid surface can cause convergence problems

Define a fillet radius to smooth sharp corners in the analytical rigid surface

Figure 3: Smoothing an analytical rigid surface.

- The rigid surface normal must always point toward the deformable surface with which it will interact. If it does not, Abaqus/Standard will detect severe overclosures at all of the nodes on the deformable surface; the simulation will probably terminate due to convergence difficulties.

  The normals for an analytical rigid surface are defined as the directions obtained by the 90° counterclockwise rotation of the vectors from the beginning to the end of each line and circular segment forming the surface (see *Figure 4*).

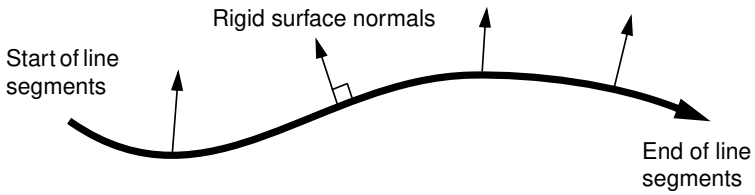


Figure 4: Normals for an analytical rigid surface.

The normals for a rigid surface created from rigid elements are defined by the faces specified when creating the surface.

Abaqus/Standard 2D example: forming a channel

This example examines the simulation of the forming of a channel in a long metal sheet.

In this section:

- *Simulation of the forming of a channel*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Job monitoring*
- *Troubleshooting Abaqus/Standard contact analyses*
- *Postprocessing*

Simulation of the forming of a channel

This simulation of the forming of a channel in a long metal sheet illustrates the use of rigid surfaces and some of the more complex techniques often required for a successful contact analysis in Abaqus/Standard.

The problem consists of a strip of deformable material, called the blank, and the tools—the punch, die, and blank holder—that contact the blank. The tools are modeled as (analytical) rigid surfaces because they are much stiffer than the blank. *Figure 1* shows the basic arrangement of the components.

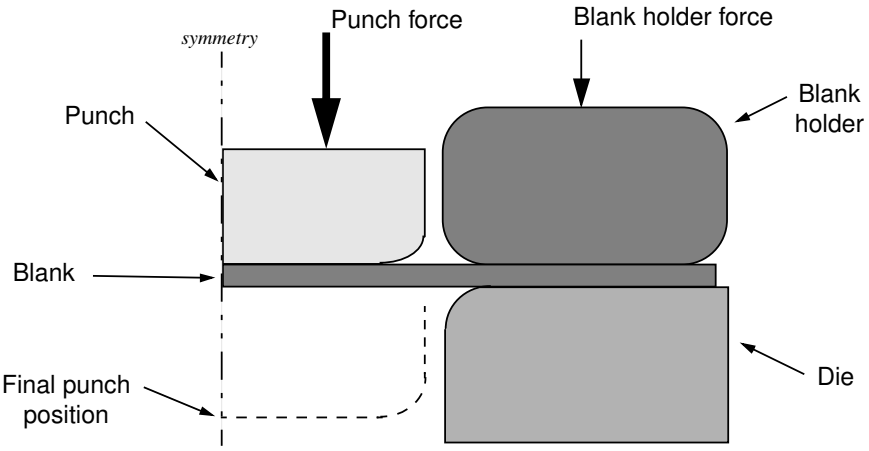


Figure 1: Forming analysis.

The blank is 1 mm thick and is squeezed between the blank holder and the die. The blank holder force is 440 kN. This force, in conjunction with the friction between the blank and blank holder and the blank and die, controls how the blank material is drawn into the die during the forming process. You have been asked to determine the forces acting on the punch during the forming process. You also must assess how well the channel is formed with these particular settings for the blank holder force and the coefficient of friction between the tools and blank.

A two-dimensional, plane strain model will be used. The assumption that there is no strain in the out-of-plane direction of the model is valid if the structure is long in this direction. Only half of the channel needs to be modeled because the forming process is symmetric about a plane along the center of the channel.

The problem can use either contact pairs or the general contact algorithm. We will use contact pairs to illustrate their use.

The dimensions of the various components are shown in *Figure 2*.

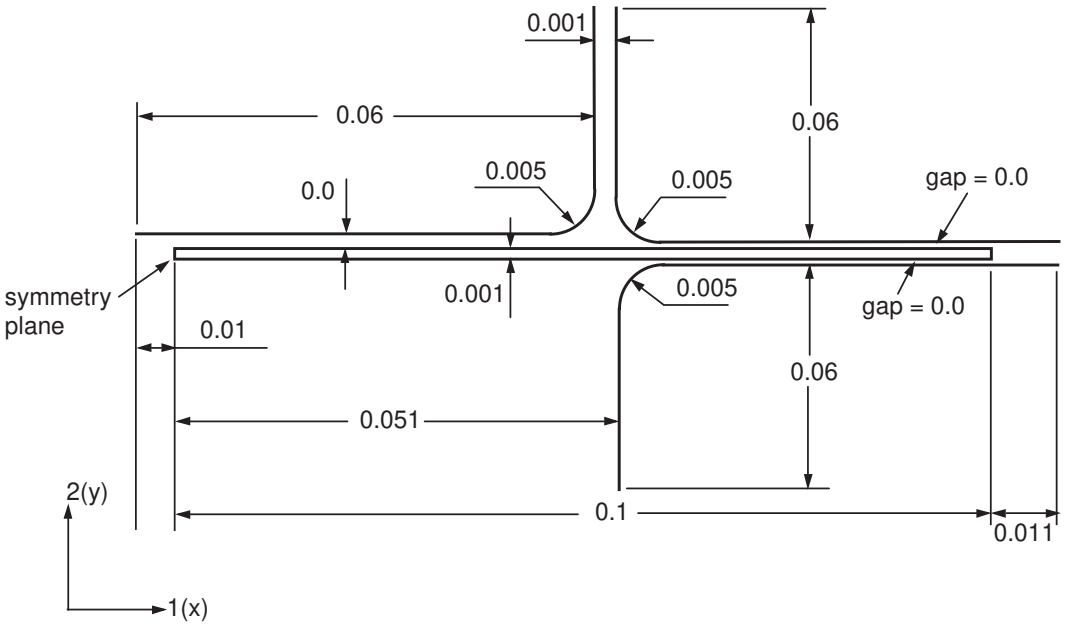


Figure 2: Dimensions, in m, of the components in the forming simulation.

Preprocessing—creating the model with Abaqus/CAE

Use Abaqus/CAE to create the model.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Forming a channel*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Forming a channel**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Part definition

Start Abaqus/CAE (if you are not already running it). You will have to create four parts: a deformable part representing the blank and three rigid parts representing the tools.

Deformable blank

Create a two-dimensional, deformable solid part with a planar shell base feature to represent the deformable blank. Use an approximate part size of `0.25`, and name the part `Blank`. To define the geometry, sketch a rectangle of arbitrary dimensions. Then, dimension the horizontal and vertical lengths of the rectangle, and edit the dimensions to define the part geometry precisely. The final sketch is shown in *Figure 1*.

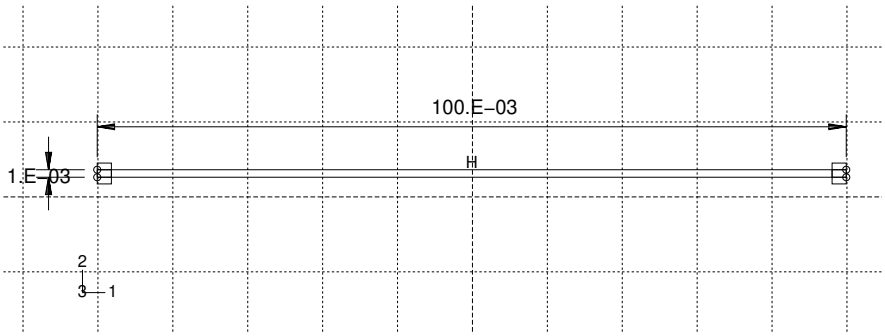


Figure 1: Sketch of the deformable blank (with grid spacing doubled).

Rigid tools

You must create a separate part for each rigid tool. Each of these parts will be created using very similar techniques so it is sufficient to consider the creation of only one of them (for example, the punch) in detail. Create a two-dimensional planar, analytical rigid part with a wire base feature to represent the rigid punch. Use an approximate part size of `0.25`, and name the part `Punch`. Using the **Create Lines** and **Create Fillet** tools, sketch the geometry of the part. Create and edit the dimensions as necessary to define the geometry precisely. The final sketch is shown in *Figure 2*.

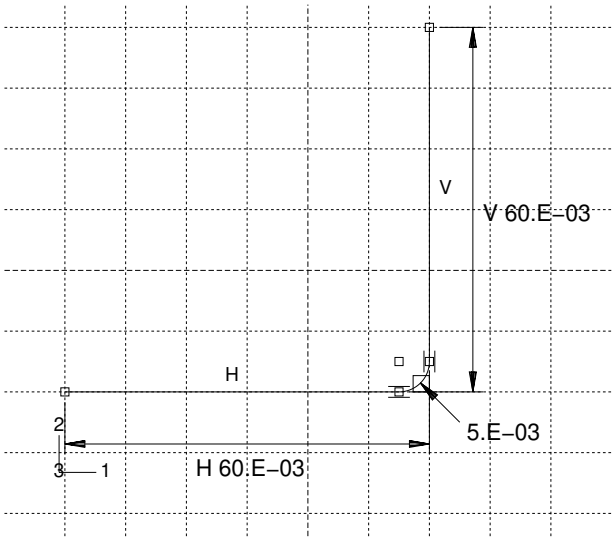


Figure 2: Sketch of the rigid punch (with grid spacing doubled).

A rigid body reference point must be created. Exit the Sketcher when you are finished defining the part geometry to return to the Part module. From the main menu bar, select **Tools->Reference Point**. In the viewport, select the point at the center of the arc as the location of the rigid body reference point.

Next, create two additional analytical rigid parts named `Holder` and `Die`, representing the blank holder and rigid die, respectively. Since the parts are mirror images of each other, the easiest way to define the geometry of the new parts is to rotate the sketch created for the punch. (The **Copy Part** tool cannot be used to mirror analytical rigid parts.) For example, edit the punch feature section sketch, and save this sketch with the name `Punch`. Then, create a part named `Holder`, and add the `Punch` sketch to the part definition. Mirror the sketch about the vertical edge. Finally, create a part named `Die`, and add the `Punch` sketch to the part definition. In this case mirror the sketch twice: first about the vertical edge and then about the horizontal edge. Be sure to create a reference point at the center of the arc on each part.

Material and section properties

The blank is made from a high-strength steel (elastic modulus of 210.0×10^9 Pa, $\nu = 0.3$). Its inelastic stress-strain behavior is tabulated in *Table 1* and shown in *Figure 3*. The material undergoes considerable work hardening as it deforms plastically. It is likely that plastic strains will be large in this analysis; therefore, hardening data are provided up to 50% plastic strain.

Table 1: Yield stress–plastic strain data.

Yield stress (Pa)	Plastic strain
400.0E6	0.0
420.0E6	2.0E–2
500.0E6	20.0E–2
600.0E6	50.0E–2

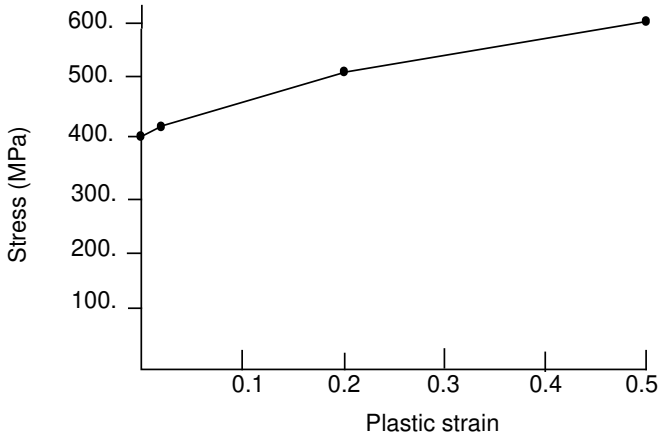


Figure 3: Yield stress vs. plastic strain.

Create a material named `Steel` with these properties. Create a homogeneous solid section named `BlankSection` that refers to the material `Steel`. Assign the section to the blank.

The blank is going to undergo significant rotation as it deforms. Reporting the values of stress and strain in a coordinate system that rotates with the blank's motion will make it much easier to interpret the results. Therefore, a local material coordinate system that is aligned initially with the global coordinate system but moves with the elements as they deform should be created. To do this, create a rectangular datum coordinate system using the **Create Datum CSYS: 3 Points** tool. From the main menu bar of the Property module, select **Assign->Material Orientation**. Select the blank as the region to which the local material orientation will be assigned, and pick the datum coordinate system in the viewport as the **CSYS** (select **Axis 3** and accept **None** for the additional rotation options).

Assembling the parts

You will now create an assembly of part instances to define the analysis model. Begin by instancing the blank. Then, instance and position the rigid tools using the techniques described below.

To instance and position the punch:

1. In the Model Tree, double-click **Instances** underneath the **Assembly** container and select `Punch` as the part to instance.

   Two-dimensional plane strain models must be defined in the global 1–2 plane. Therefore, do not rotate the parts after they have been instanced. You may, however, place the origin of the model at any convenient location. The 1-direction will be normal to the symmetry plane.

2. The bottom of the punch initially rests on top of the blank, as indicated in *Figure 2*. From the main menu bar, select **Constraint->Edge to Edge** to position the punch vertically with respect to the blank.

3. Choose the horizontal edge of the punch as the straight edge of the movable instance and the edge on the top of the blank as the straight edge of the fixed instance.

   Arrows appear on both instances. The punch will be moved so that its arrow points in the same direction as the arrow on the blank.

4. If necessary, click **Flip** in the prompt area to reverse the direction of the arrow on the punch so that both arrows point in the same direction; otherwise, the punch will be flipped. When both arrows point in the same direction, click **OK**.

5. Enter a distance of `0.0` m to specify the separation between the instances.

The punch is moved in the viewport to the specified location. Click the **Auto-fit** tool  so that the entire assembly is rescaled to fit in the viewport.

6. The vertical edge of the punch is 0.05 m from the left edge of the blank, as shown in *Figure 2*. Define another **Edge to Edge** constraint to position the punch horizontally with respect to the blank.

   Select the vertical edge of the punch as the straight edge of the movable instance and the left edge of the blank as the straight edge of the fixed instance. Flip the arrow on the punch if necessary so that both arrows point in the same direction. Enter a distance of 0.05 m to specify the separation between the edges (if the blank is positioned in the wrong direction, edit the constraint and enter –0.05 as the separation).

   Now that you have positioned the punch relative to the blank, check to make sure that the left end of the punch extends beyond the left edge of the blank. This is necessary to prevent any nodes associated with the blank from "falling off" the rigid surface associated with the punch during the contact calculations. If necessary, return to the Part module and edit the part definition to satisfy this requirement.

To instance and position the blank holder:

The procedure for instancing and positioning the holder is very similar to that used to instance and position the punch. Referring to *Figure 2*, we see that the holder is initially positioned so that its horizontal edge is offset a distance of 0.0 m from the top edge of the blank and its vertical edge is offset a distance of 0.001 m from the vertical edge of the punch. Define the necessary **Edge to Edge** constraints to position the blank holder. Remember to flip the directions of the arrows as necessary, and make sure the right end of the holder extends beyond the right edge of the blank. If necessary, return to the Part module and edit the part definition.

To instance and position the die:

The procedure for instancing and positioning the die is very similar to that used to instance and position the other tools. Referring to *Figure 2*, we see that the die is initially positioned so that its horizontal edge is offset a distance of 0.0 m from the bottom edge of the blank and its vertical edge is offset a distance of 0.0 m from the vertical edge of the holder. Define the necessary **Edge to Edge** constraints to position the die. Remember to flip the directions of the arrows as necessary, and make sure the right end of the die extends beyond the right edge of the blank. If necessary, return to the Part module and edit the part definition.

The final assembly is shown in *Figure 4*.

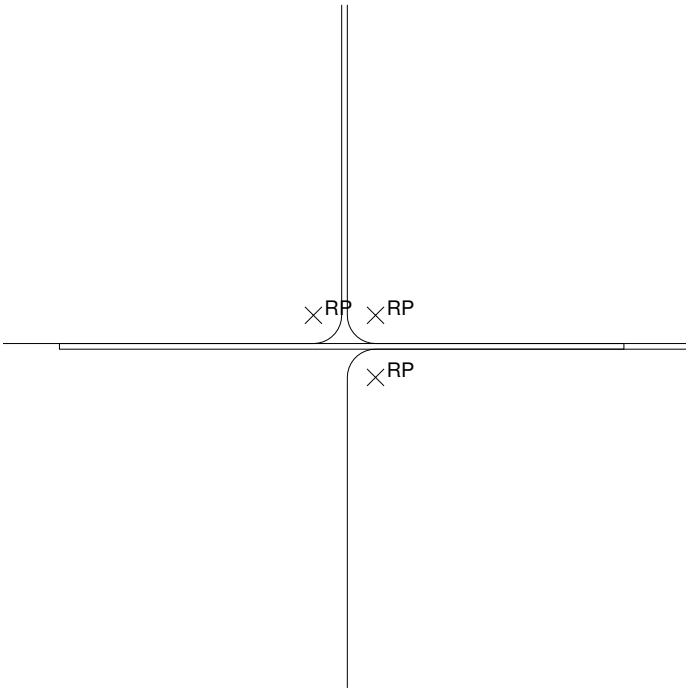


Figure 4: Model assembly.

Geometry sets

At this point it is convenient to create the geometry sets that will be used to specify loads and boundary conditions and to restrict data output. Four sets should be created: one at each rigid body reference point, and one at the symmetry plane of the blank.

To create geometry sets:

1. Double-click the **Sets** item underneath the **Assembly** container to create the following geometry sets:
   - `RefPunch` at the punch rigid body reference point.
   - `RefHolder` at the holder rigid body reference point.
   - `RefDie` at the die rigid body reference point.
   - `Center` at the left vertical edge (symmetry plane) of the blank.

Defining steps and output requests

There are two major sources of difficulty in Abaqus/Standard contact analyses: rigid body motion of the components before contact conditions constrain them and sudden changes in contact conditions, which lead to severe discontinuity iterations as Abaqus/Standard tries to establish the correct condition of all contact surfaces. Therefore, wherever possible, take precautions to avoid these situations.

Removing rigid body motion is not particularly difficult. Simply ensure that there are enough constraints to prevent all rigid body motions of all the components in the model. This may mean using boundary conditions initially to get the components into contact, instead of applying loads directly. Using this approach may require more steps than originally anticipated, but the solution of the problem should proceed more smoothly.

Alternatively, contact controls can be used to stabilize rigid body motion automatically. With this approach Abaqus/Standard applies viscous damping to the secondary nodes of the contact pair. Care must be taken, however, to ensure that the viscous damping does not significantly alter the physics of the problem, as will be the case if the dissipated stabilization energy and contact damping stresses are sufficiently small.

The channel-forming simulation will consist of two steps. Since the simulation involves material, geometric, and boundary nonlinearities, general steps must be used. In addition, the forming process is quasi-static; thus, we can ignore inertia effects throughout the simulation. Rather than use additional steps to establish firm contact, contact stabilization as described above will be used. A brief summary of each step (including the details of its purpose, definition, and associated output requests) is given below. However, the details concerning how the loads and boundary conditions are applied are discussed later.

Step 1

The magnitude of the blank holder force is a controlling factor in many forming processes; therefore, it needs to be introduced as a variable load in the analysis. In this step the blank holder force will be applied.

Given the quasi-static nature of the problem and the fact that nonlinear response will be considered, create a static, general step named `Holder force` after the `Initial` step. Enter the following description for the step, `Apply holder force`; and include the effects of geometric nonlinearity. Set the initial time increment to `0.05` and the total time period to `1.0`. Specify that the preselected field output be written every 20 increments for this step. In addition, request that the vertical reaction force and displacement (**RF2** and **U2**) at the punch reference point (geometry set `RefPunch`) be written every increment as history data. In addition, write contact diagnostics to the message file (**Output->Diagnostic Print**).

Step 2

In the second and final step the punch will be moved down to complete the forming operation.

Create a static, general step named `Move punch`, and insert it after the `Holder force` step. Enter the following description for the step: `Apply punch stroke`. Because of the frictional sliding, the changing contact conditions, and the inelastic material behavior, there is significant nonlinearity in this step; therefore, set the maximum number of increments to a large value (for example, `1000`). Set the initial time increment to `0.05` and the total time period to `1.0`. Your output requests from the previous step will be propagated to this step. In addition, request that the restart file be written every `200` increments for this step.

Monitoring the value of a degree of freedom

You can request that Abaqus monitor the value of a degree of freedom at one selected point. The value of the degree of freedom is shown in the **Job Monitor** and is written at every increment to the status (`.sta`) file and at specific increments during the course of an analysis to the message (`.msg`) file. In addition, a plot of the degree of freedom value over time appears in a new viewport that is generated automatically when you submit the analysis. You can use this information to monitor the progress of the solution.

In this model you will monitor the vertical displacement (degree of freedom 2) of the punch's reference node throughout each step. Before proceeding, make the first analysis step (`Holder force`) active by selecting it from the **Step** list located in the context bar. The monitor definition applied for this step will be propagated automatically to the subsequent step.

To select a degree of freedom to monitor:

1. From the main menu bar of the Step module, select **Output->DOF Monitor**.

   The **DOF Monitor** dialog box appears.
2. Toggle on **Monitor a degree of freedom throughout the analysis**.

3. Click  to select the region. In the prompt area, click **Points**. In the **Region Selection** dialog box that appears, select **RefPunch**; and click **Continue**.
4. In the **Degree of freedom** text field, enter `2`.
5. Accept the default frequency (every increment) at which this information will be written to the message file.
6. Click **OK** to exit the **DOF Monitor** dialog box.

Defining contact interactions

Contact must be defined between the top of the blank and the punch, the top of the blank and the blank holder, and the bottom of the blank and the die. The rigid surface must be the main surface in each of these contact interactions. Each contact interaction must refer to a contact interaction property that governs the interaction behavior.

In this example we assume that the friction coefficient is zero between the blank and the punch. The friction coefficient between the blank and the other two tools is assumed to be 0.1. Therefore, two contact interaction properties must be defined: one with friction and one without.

Define the following surfaces: `BlankTop` on the top edge of the blank; `BlankBot` on the bottom edge of the blank; `DieSurf` on the side of the die that faces the blank; `HolderSurf` on the side of the holder that faces the blank; and `PunchSurf` on the side of the punch that faces the blank.



Tip: To facilitate your selections, you can selectively hide part instances using the Model Tree: expand the **Instances** container, highlight the part instances that you want to hide, and click mouse button 3. From the menu that appears, select **Hide**. To restore the visibility of the part instances, repeat the procedure, and select **Show** from the menu.

Now define two contact interaction properties. (In the Model Tree, double-click the **Interaction Properties** container to create a contact property.) Name the first one `NoFric`; since frictionless contact is the default in Abaqus, accept the default property settings for the tangential behavior (select **Mechanical->Tangential Behavior** in the **Edit Contact Property** dialog box). The second property should be named `Fric`. For this property use the **Penalty** friction formulation with a friction coefficient of `0.1`.

To alleviate convergence difficulties that may arise due to the changing contact states (in particular for contact between the punch and the blank), create contact controls to invoke automatic contact stabilization. Scale down the default damping factor by a factor of 1,000 to minimize the effects of stabilization on the solution. The procedure is described next.

To define contact controls:

1. In the Model Tree, double-click the **Contact Controls** container to define the contact controls.

   The **Create Contact Controls** dialog box appears.
2. Name the control `stabilize`. Select **Abaqus/Standard contact controls**, and click **Continue**.
3. In the **Stabilization** tabbed page of the **Edit Contact Controls** dialog box, toggle on **Automatic stabilization** and set the **Factor** to `0.001`.
4. Click **OK** to exit the **Edit Contact Controls** dialog box.

Finally, define the interactions between the surfaces and refer to the appropriate contact interaction property for each definition. (In the Model Tree, double-click the **Interactions** container to define a contact interaction.) In all cases define the interactions in the `Initial` step and use the **Surface-to-surface contact (Standard)** type. When defining the interactions, use the default finite-sliding formulation. The following interactions should be defined:

- `Die-Blank` between surfaces `DieSurf` (main) and `BlankBot` (secondary) referring to the `Fric` contact interaction property. Accept the default contact controls.
- `Holder-Blank` between surfaces `HolderSurf` (main) and `BlankTop` (secondary) referring to the `Fric` contact interaction property. Accept the default contact controls.
- `Punch-Blank` between surfaces `PunchSurf` (main) and `BlankTop` (secondary) referring to the `NoFric` contact interaction property. Using the **Interaction Manager**, edit this interaction to assign the nondefault contact controls defined earlier (`stabilize`) in the second analysis step (`Move punch`).

Boundary conditions and loading for Step 1

In this step contact will be established between the blank holder and the blank while the punch and die are held fixed.

Constrain the blank holder in degrees of freedom 1 and 6, where degree of freedom 6 is the rotation in the plane of the model; constrain the punch and die completely. All of the boundary conditions for the rigid surfaces are applied to their respective rigid body reference nodes. Apply symmetric boundary constraints on the region of the blank lying on the symmetry plane (geometry set `Center`).

Table 2 summarizes the boundary conditions applied in this step.

Table 2: Summary of boundary conditions applied in Step 1.

BC Name	Geometry Set	BCs
CenterBC	Center	XSYMM
RefDieBC	RefDie	U1 = U2 = UR3 = 0.0
RefHolderBC	RefHolder	U1 = UR3 = 0.0
RefPunchBC	RefPunch	U1 = U2 = UR3 = 0.0

To apply the blank holder force, create a mechanical concentrated force named `RefHolderForce`. Recall that in this simulation the required blank holder force is 440 kN. Thus, apply the load to set `RefHolder`, and specify a magnitude of `-440.E3` for **CF2**.

Boundary conditions for Step 2

In this step move the punch down to complete the forming operation. Using the **Boundary Condition Manager**, edit the **RefPunchBC** boundary condition to specify a value of −0.030 for **U2**, which represents the total displacement of the punch.

Before continuing, change the name of your model to `Standard`.

Mesh creation and job definition

You should consider the type of element you will use before you design your mesh. When choosing an element type, you must consider several aspects of your model such as the model's geometry, the type of deformation that will be seen, the loads being applied, etc. The following points are important to consider in this simulation:

- The contact between surfaces. Whenever possible, first-order elements (with the exception of tetrahedral elements) should be used for contact simulations. When using tetrahedral elements, second-order tetrahedral elements should be used for contact simulations (use either the regular or modified form for the surface-to-surface discretization, and use the modified form for the node-to-surface discretization).

- Significant bending of the blank is expected under the applied loading. Fully integrated first-order elements exhibit shear locking when subjected to bending deformation. Therefore, either reduced-integration or incompatible mode elements should be used.

Either incompatible mode or reduced-integration elements are suitable for this analysis. In this analysis you will use reduced-integration elements with enhanced hourglass control. Reduced-integration elements help decrease the analysis time, and enhanced hourglass control reduces the possibility of hourglassing in the model. Mesh the blank with CPE4R elements using enhanced hourglass control (see *Figure 5*).

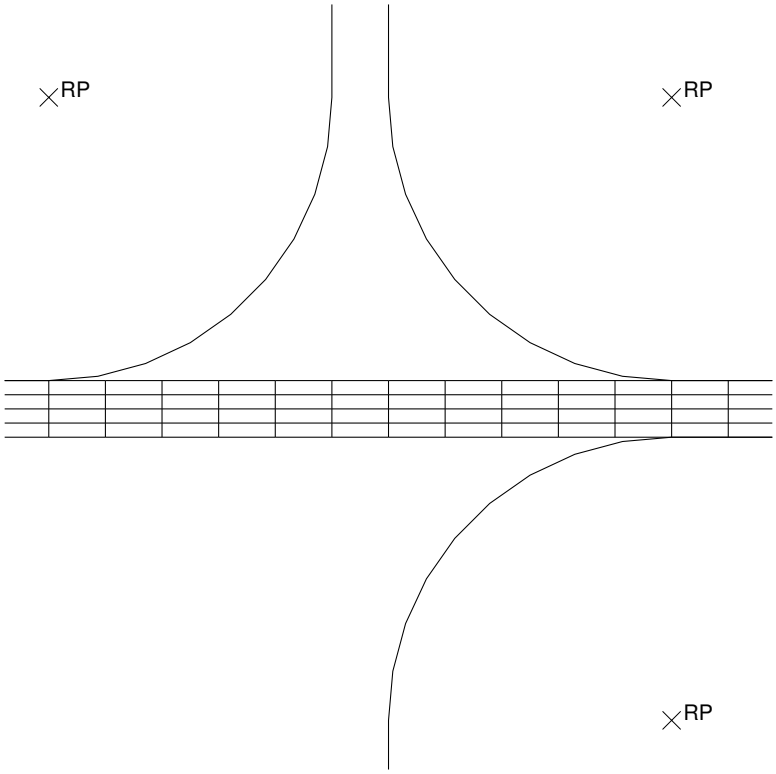


Figure 5: Mesh for the channel forming analysis.

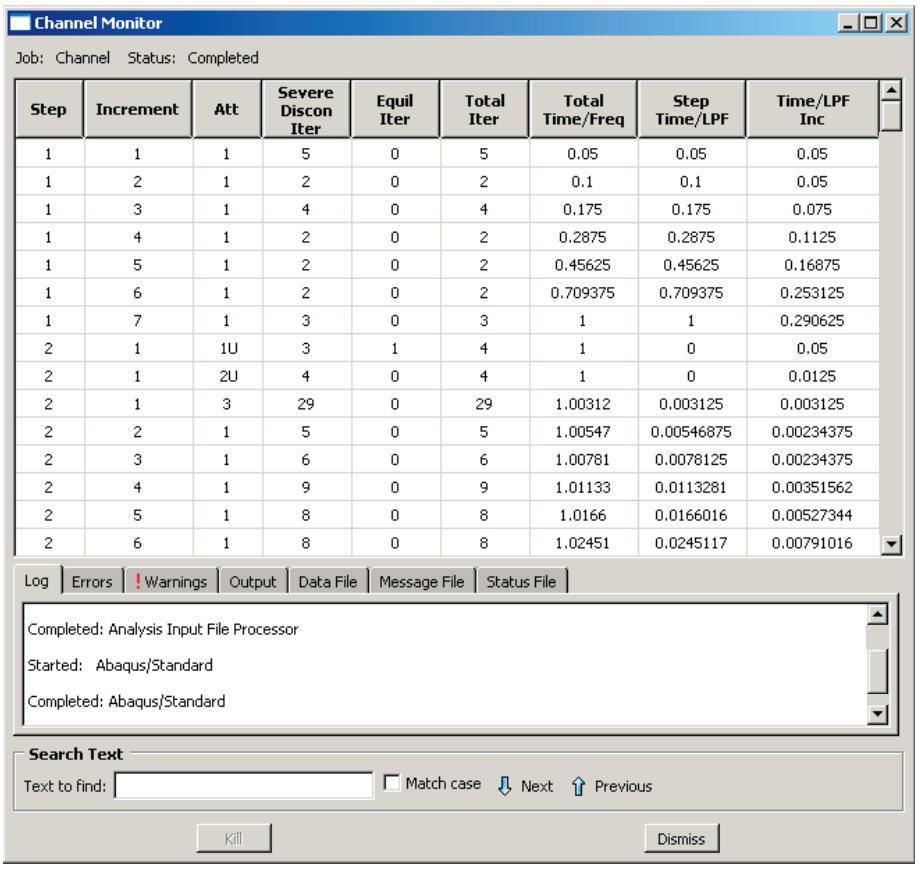
Seed the edges of the blank by specifying the number of elements along each edge. Specify `100` elements along the horizontal edges of the blank and `4` elements along each vertical edge of the blank. The tools have been modeled with analytical rigid surfaces so they need not be meshed. However, if the tools had been modeled with discrete rigid elements, the mesh would have to be sufficiently refined to avoid contact convergence difficulties. For example, if the die were modeled with R2D2 elements, the curved corner should be modeled with at least 20 elements. This would create a sufficiently smooth surface that would capture the corner geometry accurately. Always use a sufficient number of elements to model such curves when using discrete rigid elements.

Create a job named `Channel`. Give the job the following description: `Analysis of the forming of a channel`. Save your model to a model database file, and submit the job for analysis. Monitor the solution progress, correct any modeling errors that are detected, and investigate the cause of any warning messages.

Once the analysis is underway, an *X*–*Y* plot of the values of the degree of freedom that you selected to monitor (the punch's vertical displacement) appears in a separate viewport. From the main menu bar, select **Viewport->DOF Monitor: Channel** to follow the progression of the punch's displacement in the 2-direction over time as the analysis runs.

Job monitoring

This analysis should take approximately 180 increments to complete. The top of the **Job Monitor** is shown in *Figure 1*.



Step	Increment	Att	Severe Discon Iter	Equil Iter	Total Iter	Total Time/Freq	Step Time/LPF	Time/LPF Inc
1	1	1	5	0	5	0.05	0.05	0.05
1	2	1	2	0	2	0.1	0.1	0.05
1	3	1	4	0	4	0.175	0.175	0.075
1	4	1	2	0	2	0.2875	0.2875	0.1125
1	5	1	2	0	2	0.45625	0.45625	0.16875
1	6	1	2	0	2	0.709375	0.709375	0.253125
1	7	1	3	0	3	1	1	0.290625
2	1	1U	3	1	4	1	0	0.05
2	1	2U	4	0	4	1	0	0.0125
2	1	3	29	0	29	1.00312	0.003125	0.003125
2	2	1	5	0	5	1.00547	0.00546875	0.00234375
2	3	1	6	0	6	1.00781	0.0078125	0.00234375
2	4	1	9	0	9	1.01133	0.0113281	0.00351562
2	5	1	8	0	8	1.0166	0.0166016	0.00527344
2	6	1	8	0	8	1.02451	0.0245117	0.00791016

Figure 1: Top of the Job Monitor: channel forming analysis.

The value of the punch displacement appears in the **Output** tabbed page. This simulation contains many severe discontinuity iterations. Abaqus/Standard has a difficult time determining the contact state in the first increment of Step 2. It needs three attempts before it finds the proper configuration of the `PunchSurf` and `BlankTop` surfaces and achieves equilibrium. After this difficult start, Abaqus/Standard quickly increases the increment size to a more reasonable value. The end of the **Job Monitor** is shown in *Figure 2*.

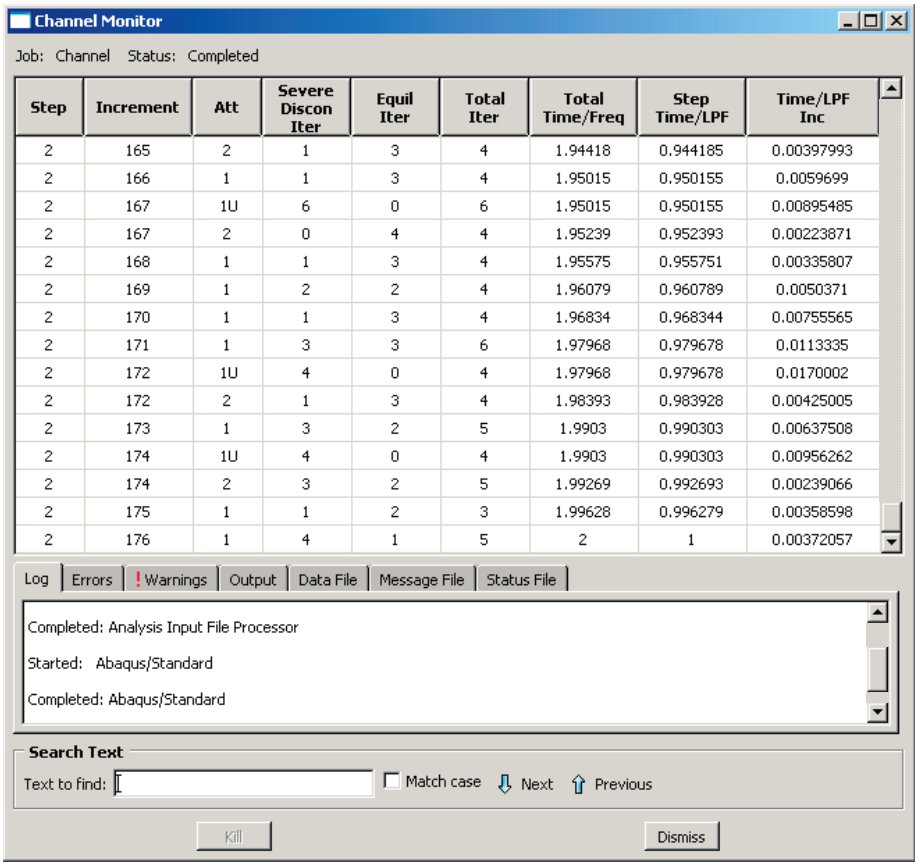


Figure 2: Bottom of the Job Monitor: channel forming analysis.

Troubleshooting Abaqus/Standard contact analyses

Contact analyses are generally more difficult to complete than just about any other type of simulation in Abaqus/Standard. Therefore, it is important to understand all of the options available to help you with contact analyses.

If a contact analysis runs into difficulty, the first thing to check is whether the contact surfaces are defined correctly. The easiest way to do this is to run a **datacheck** analysis and plot the surface normals in the Visualization module. You can plot all of the normals, for both surfaces and structural elements, on either the deformed or the undeformed plots. Use the **Normals** options in the **Common Plot Options** dialog box to do this, and confirm that the surface normals are in the correct directions.

Abaqus/Standard may still have some problems with contact simulations, even when the contact surfaces are all defined correctly. One reason for these problems may be the default convergence tolerances and limits on the number of iterations: they are quite rigorous. In contact analyses it is sometimes better to allow Abaqus/Standard to iterate a few more times rather than abandon the increment and try again. This is why Abaqus/Standard makes the distinction between severe discontinuity iterations and equilibrium iterations during the simulation.

The diagnostic contact information is essential for almost every contact analysis. This information can be vital for spotting mistakes or problems. For example, chattering can be spotted because the same secondary node will be seen to be involved in all of the severe discontinuity iterations. If you see this, you will have to modify the mesh in the region around that node or add constraints to the model. Contact diagnostic information can also identify regions where only a single secondary node is interacting with a surface. This is a very unstable situation and can cause convergence problems. Again, you should modify the model to increase the number of elements in such regions.

Contact diagnostics

To illustrate how to interpret the contact diagnostic information in Abaqus/CAE, consider the iterations in the seventh increment of the second step. This increment is one in which severe discontinuity iterations are required. Abaqus/Standard requires three iterations to establish the correct contact conditions in the model; i.e., whether or not the punch was contacting the blank. The fourth and fifth iterations do not produce any changes in the model's contact state but do not achieve equilibrium. One additional iteration is required to converge on static equilibrium. Thus, once Abaqus/Standard determines the correct contact state, it can easily find the equilibrium solution.

To further investigate the behavior of the model in this increment, look at the visual diagnostic information available in Abaqus/CAE. The diagnostic information written to the output database file provides detailed information about the changes in the model's contact conditions. For example, the node number and location in the model of every secondary node whose contact status changes in a severe discontinuity iteration, as well as the contact interaction to which it belongs, can be obtained using the visual diagnostics tool.

Enter the Visualization module, and open the file `Channel.odb` to look at the contact diagnostics information. In the first severe discontinuity iteration of the second step (increment 7, attempt 1), four nodes on the blank experience contact openings, indicating that their assumed contact state is incompatible. This incompatibility can be seen in the **Contact** tabbed page of the **Job Diagnostics** dialog box (see *Figure 1*). To see where the nodes are located on the model, toggle on **Highlight selections in viewport**.

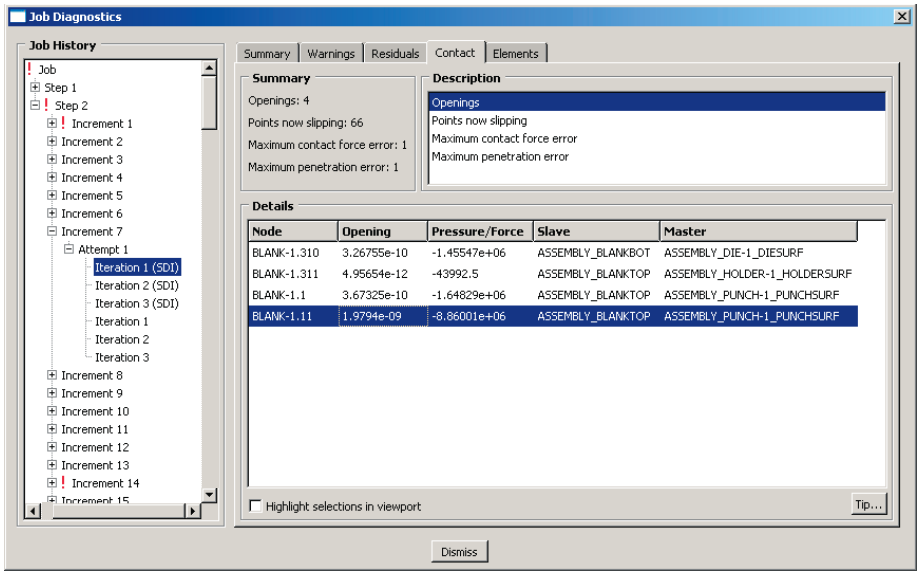


Figure 1: Contact openings in the first severe discontinuity iteration.

Since neither the contact state nor the equilibrium checks pass in this iteration, Abaqus/Standard removes the contact constraints from these nodes and performs another iteration. After two additional iterations Abaqus/Standard detects no changes in the contact state. The solution in the fourth and fifth iterations do not satisfy the force residual tolerance check, so another iteration is performed. This time, not only is the contact state converged, but the force residual tolerance check is satisfied and the displacement correction is acceptable relative to the largest displacement increment, as shown in *Figure 2*. Thus, the third equilibrium iteration produces a converged solution for this increment.

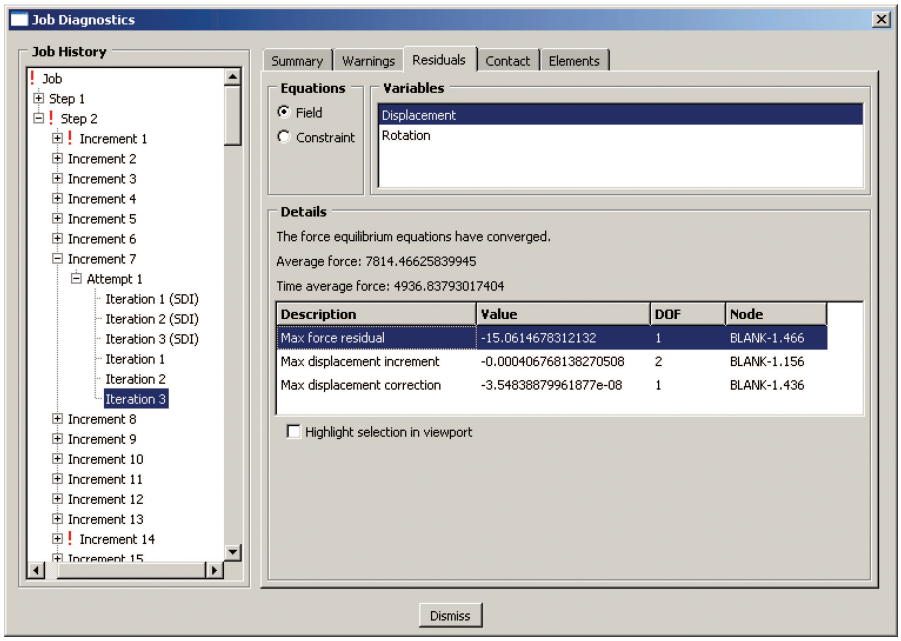


Figure 2: Converged equilibrium iteration.

Postprocessing

In the Visualization module, examine the deformation of the blank.

Deformed model shape and contour plots

The basic result of this simulation is the deformation of the blank and the plastic strain caused by the forming process. We can plot the deformed model shape and the plastic strain, as described below.

To plot the deformed model shape:

1. Plot the deformed model shape. You can remove the die and the punch from the display and visualize just the blank.
2. In the Results Tree, expand the **Instances** container underneath the output database file named `Channel.odb`.
3. From the list of available part instances, select **BLANK-1**. Click mouse button 3, and select **Replace** from the menu that appears to replace the current display group with the selected elements. Click , if necessary, to fit the model in the viewport.

   The resulting plot is shown in *Figure 1*.

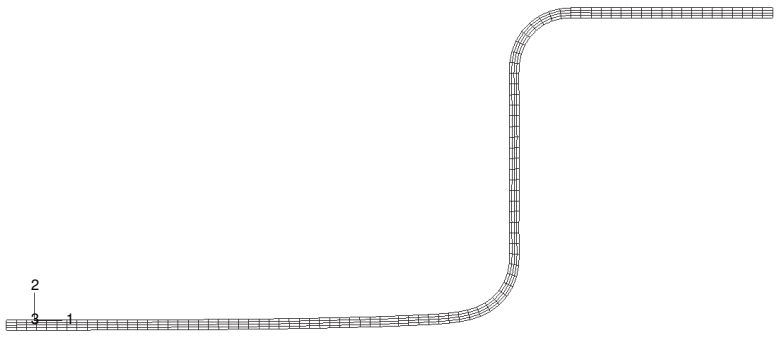


Figure 1: Deformed shape of blank at the end of Step 2.

To plot the contours of equivalent plastic strain:

1. From the main menu bar, select **Plot->Contours->On Deformed Shape**; or click the  tool from the toolbox to display contours of Mises stress.
2. Open the **Contour Plot Options** dialog box.
3. Drag the **Contour Intervals** slider to change the number of contour intervals to **7**.
4. Click **OK** to apply these settings.
5. Select **Primary** from the list of variable types on the left side of the **Field Output** toolbar, and select **PEEQ** from the list of output variables.

   PEEQ is an integrated measure of plastic strain. A nonintegrated measure of plastic strain is PEMAG. PEEQ and PEMAG are equal for proportional loading.
6. Use the  tool to zoom into any region of interest in the blank, as shown in *Figure 2*.

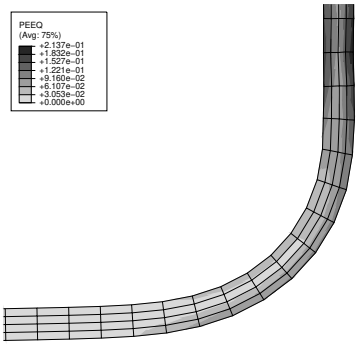


Figure 2: Contours of the scalar plastic strain variable PEEQ in one corner of the blank.

The maximum plastic strain is approximately 21%. Compare this with the failure strain of the material to determine if the material will tear during the forming process.

History plots of the reaction forces on the blank and punch

The solid line in *Figure 3* shows the variation of the reaction force RF2 at the punch's rigid body reference node.

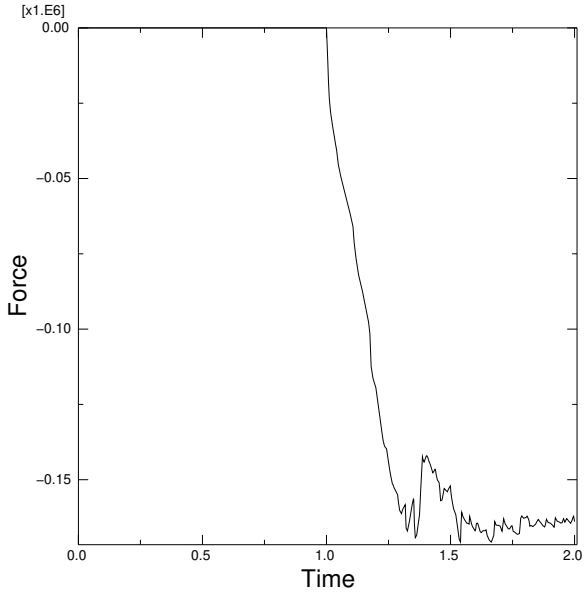


Figure 3: Force on punch.

To create a history plot of the reaction force:

1. In the Results Tree, expand the **History Output** container. Double-click `Reaction force: RF1 PI: PUNCH-1 Node xxx in NSET REFPUNCH`.

   A history plot of the reaction force in the 1-direction appears.
2. Open the **Axis Options** dialog box to label the axes.
3. Switch to the **Title** tabbed page.
4. Specify `Reaction Force - RF2` as the *Y*-axis label, and `Total Time` as the *X*-axis label.

5. Click **Dismiss** to close the dialog box.

The punch force, shown in *Figure 3*, rapidly increases to about 160 kN during Step 2, which runs from a total time of 1.0 to 2.0.

History plot of the stabilization and internal energies

It is important to verify that the presence of contact stabilization does not significantly alter the physics of the problem. One way to assess this requirement is to compare the energy dissipation due to stabilization (ALLSD) against the internal energy of the structure (ALLIE). Ideally the amount of stabilization energy should be a small fraction of the internal energy. *Figure 4* shows the variation of the stabilization and internal energies. It is clear that the dissipated stabilization energy is indeed small.

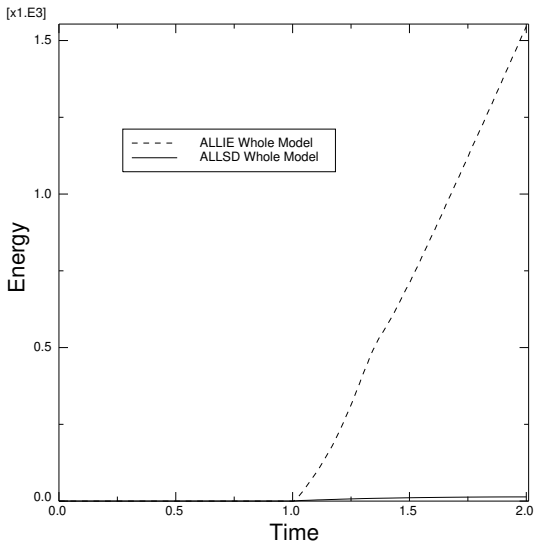


Figure 4: Stabilization and internal energies.

Plotting contours on surfaces

Abaqus/CAE includes a number of features designed specifically for postprocessing contact analyses. Within the Visualization module, the **Display Group** feature can be used to collect surfaces into display groups, similar to element and node sets.

To display contact surface normal vectors:

1. Plot the undeformed model shape.
2. In the Results Tree, expand the **Surface Sets** container. Select the surfaces named `BLANKTOP` and `PUNCH-1.PUNCHSURF`. Click mouse button 3, and select **Replace** from the menu that appears.
3. Using the **Common Plot Options** dialog box, turn on the display of the normal vectors (**On surfaces**) and set the length of the vector arrows to **Short**.
4. Use the  tool, if necessary, to zoom into any region of interest, as shown in *Figure 5*.

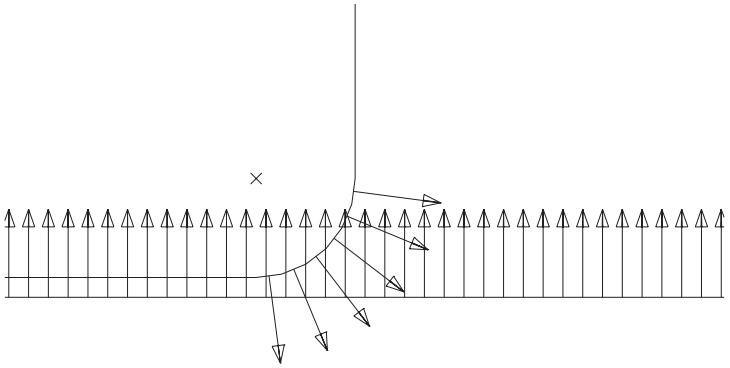


Figure 5: Surface normals.

To contour the contact pressure:

1. Plot the contours of plastic strain again.
2. From the list of variable types on the left side of the **Field Output** toolbar, select **Primary**, if it is not already selected.
3. From the list of output variables in the center of the toolbar, select `CPRESS`.
4. Remove the `PUNCH-1.PUNCHSURF` surface from your display group.

   To visualize contours of surface-based variables better in two-dimensional models, you can extrude the plane strain elements to construct the equivalent three-dimensional view. You can sweep axisymmetric elements in a similar fashion.
5. From the main menu bar, select **View->ODB Display Options**.

   The **ODB Display Options** dialog box appears.
6. Select the **Sweep/Extrude** tab to access the **Sweep/Extrude** options.
7. In the **Extrude** region of the dialog box, toggle on **Extrude elements**; and set the **Depth** to `0.05` to extrude the model for the purpose of displaying contours.
8. Click **OK** to apply these settings.

Rotate the model using the  tool to display the model from a suitable view, such as the one shown in *Figure 6*.

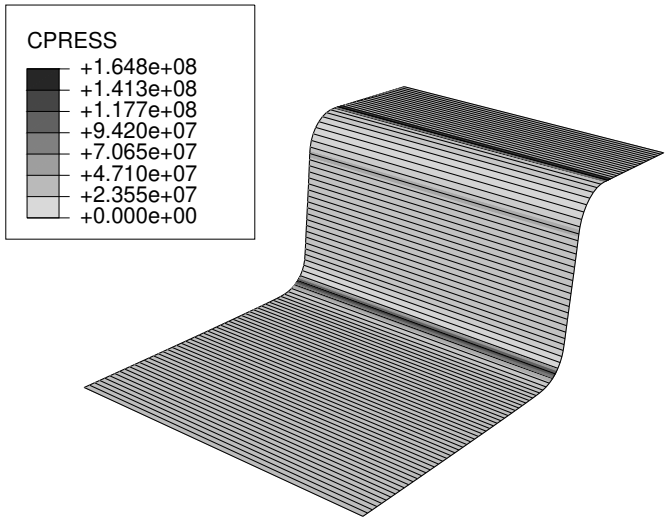


Figure 6: Contact pressure.

General contact in Abaqus/Standard

You can specify general contact in Abaqus/Standard.

In the channel forming example in *Abaqus/Standard 2D example: forming a channel*, contact interaction is defined using the contact pairs algorithm, which requires you to explicitly define the surfaces that may potentially come into contact. As an alternative, you can specify contact in an Abaqus/Standard analysis by using the general contact algorithm. The contact interaction domain, contact properties, and surface attributes are specified independently for general contact, offering a more flexible way to add detail incrementally to a model. The simple interface for specifying general contact allows for a highly automated contact definition; however, it is also possible to define contact with the general contact interface to mimic traditional contact pairs. Conversely, specifying self-contact of a surface spanning multiple bodies with the contact pair user interface (if the surface-to-surface formulation is used) mimics the highly automated approach often used for general contact.

In Abaqus/Standard traditional pairwise specifications of contact interactions will often result in more efficient analyses as compared to an all-inclusive self-contact approach to defining contact. Therefore, there is often a trade-off between ease of defining contact and analysis performance. Abaqus/CAE provides a contact detection tool that greatly simplifies the process of creating traditional contact pairs for Abaqus/Standard (see *Understanding contact and constraint detection*).

Abaqus/Standard 3D example: shearing of a lap joint

This example examines the shearing of a lap joint.

In this section:

- *Simulation of the shearing of a lap joint*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*

Simulation of the shearing of a lap joint

This simulation of the shearing of a lap joint illustrates the use of general contact in Abaqus/Standard.

The model consists of two overlapping aluminum plates that are connected with a titanium rivet. The left end of the bottom plate is fixed, and the force is applied to the right end of the top plate to shear the joint. *Figure 1* shows the basic arrangement of the components. Because of symmetry, only half of the joint is modeled to reduce computational cost. Frictional contact is assumed.

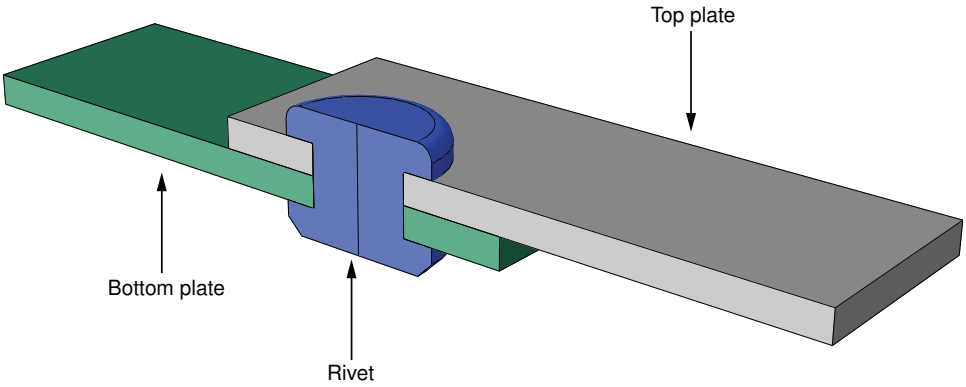


Figure 1: Lap joint analysis.

Preprocessing—creating the model with Abaqus/CAE

Use Abaqus/CAE to create the model.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Shearing of a lap joint*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**, highlight **Lap joint**, and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Part definition

Start Abaqus/CAE (if you are not already running it). You will have to create two parts: one representing the plate and one representing the rivet.

Plate

Create a three-dimensional, deformable solid part with an extruded base feature to represent the plate. Use an approximate part size of `100.0`, and name the part `plate`. Begin by sketching a rectangle of arbitrary dimensions. Then, dimension it so that the horizontal length is `30` and the vertical length is `10`, as shown in *Figure 1*.

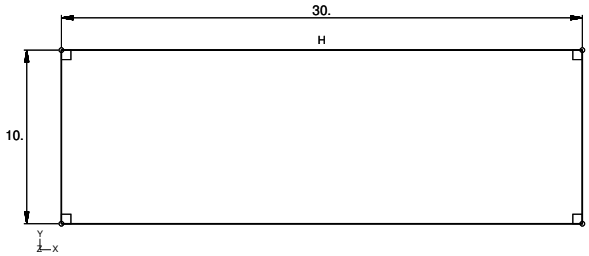


Figure 1: Sketch of the plate.

Extrude the part a distance of `1.5`.

Use the **Create Cut: Extrude** tool to cut out the circular region corresponding to the bolt hole. Select the front face of the plate as the sketch plane and the right edge of the face as the edge that will appear vertical and on the right in the sketch. Sketch the bolt hole as shown in *Figure 2*.

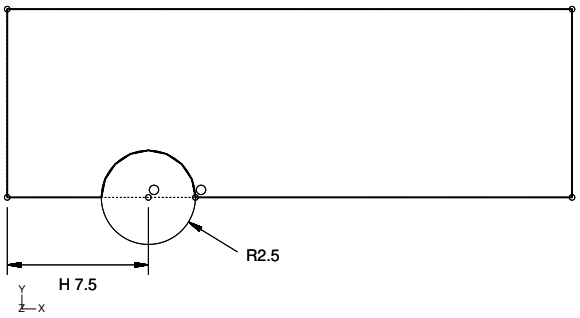


Figure 2: Sketch of the bolt hole.

Extrude the cut through the entire part.

The final shape of the plate appears as shown in *Figure 3*.

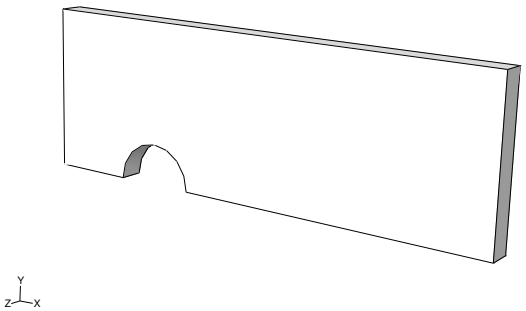


Figure 3: Final plate geometry.

Rivet

Create a three-dimensional, deformable solid part with a revolved base feature to represent the rivet. Use an approximate part size of `20.0`, and name the part `rivet`. Using the **Create Lines** tool, create a rough sketch of the rivet geometry, as shown in *Figure 4*. Use dimensions and equal length constraints as necessary to refine the sketch, as shown in *Figure 4*. Revolve the part `180` degrees.

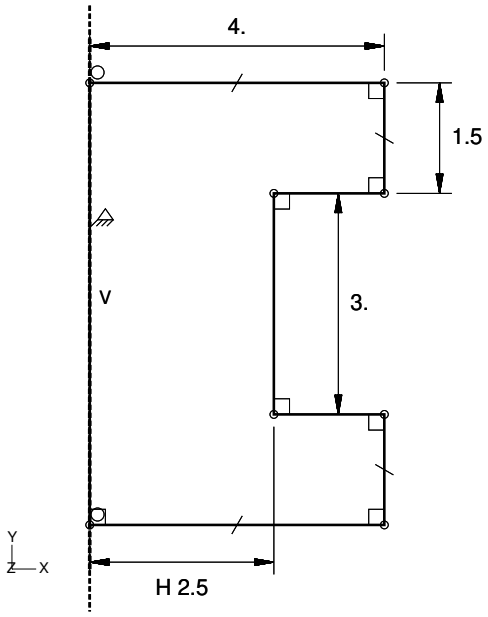


Figure 4: Base sketch of the rivet.

Edit the base part to include a fillet at the top outer edge and a chamfer at the bottom outer edge. Use a radius of `0.75` for the fillet and a length of `0.75` for the chamfer. The final part geometry is shown in *Figure 5*.

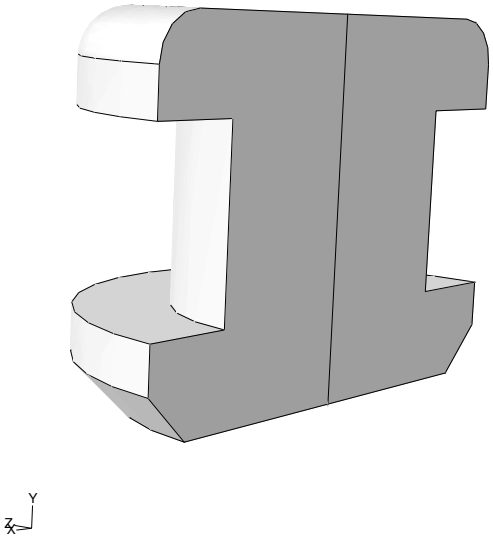


Figure 5: Final rivet geometry.

Material and section properties

The plates are made from aluminum; the stress-strain behavior is shown in *Figure 6*. The rivet is made from titanium; the stress-strain behavior is shown in *Figure 7*.

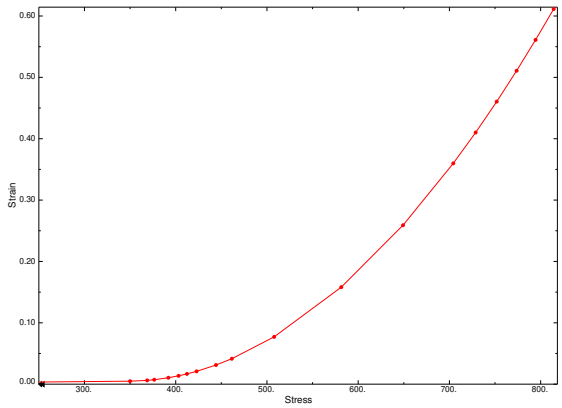


Figure 6: Aluminum stress-strain curve.

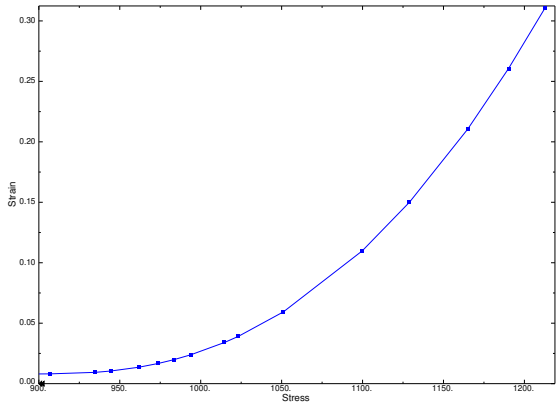


Figure 7: Titanium stress-strain curve.

The stress-strain data for the aluminum and titanium materials are provided in text files named `lap-joint-alum.txt` and `lap-joint-titanium.txt`, respectively. Enter the following command at the operating system prompt to use the Abaqus **fetch** utility to copy these files to your local directory:

```
abaqus fetch job=lap*.txt
```

Rather than convert the stress-strain data and define the material properties manually, you will use the material calibration capability to define the material properties.

To calibrate a material:

1. In the Model Tree, double-click **Calibrations**.
2. Name the calibration `aluminum`, and click **OK**.
3. Expand the **Calibrations** container and then expand the **aluminum** item.
4. Double-click **Data Sets**.
5. In the **Create Data Set** dialog box, enter `Al` as the name and click **Import Data Set**.
6. In the **Read Data From Text File** dialog box, click  and choose the file named `lap-joint-alum.txt`.
7. In the **Properties** region of this dialog box, specify that strain values will be read from field 2 and stress values from field 1.
8. From the **Data Set Form** options, select **True** to indicate that the data you are importing are in true form.

9. Click **OK** to close the **Read Data From Text File** dialog box.
10. Click **OK** to close the **Create Data Set** dialog box.
11. In the Model Tree, double-click **Behaviors**.
12. Name the behavior `Al-elastic-plastic`, choose **Elastic Plastic Isotropic** as the type, and click **Continue**.
13. In the **Edit Behavior** dialog box, choose **Al** as the data set for the elastic-plastic data.
14. Enter `0.00488, 350.0` in the text field to define the yield point (alternatively, you could select the point directly in the viewport).
15. Drag the **Plastic points** slider midway between **Min** and **Max** to generate plastic data points.
16. Enter a Poisson's ratio of `0.33`.
17. At the bottom of the dialog box, click  to create an empty material named `aluminum` (simply click **OK** in the material editor after entering the name).
18. In the **Edit Behavior** dialog box, choose **aluminum** from the **Material** drop-down list.
19. Click **OK** to add the properties to the material named **aluminum**.
20. In the Model Tree, expand the **Materials** container and examine the contents of the material model. You will note that both elastic and plastic properties have been defined. If you wish to change the number of plastic points or modify the yield point, simply return to the **Edit Behavior** dialog box, make the necessary changes, select the name of the material to which the properties will be applied, and click **OK**. The contents of the material model are updated automatically.
21. Following the same procedure, create a material model named `titanium`. The file containing the stress-strain data is named `lap-joint-titanium.txt`; the yield point is `0.0081, 907.0`; and Poisson's ratio is equal to `0.34`.

Create a homogeneous solid section named `plate` that refers to the material `aluminum`. Assign the section to the plate.

Create a homogeneous solid section named `rivet` that refers to the material `titanium`. Assign the section to the rivet.

Assembling the parts

You will now create an assembly of part instances to define the analysis model. The assembly consists of two dependent instances of the plate and a single dependent instance of the rivet. The first plate instance is the top plate of the assembly; the second plate instance is the bottom plate of the assembly.

To instance and position the plates:

1. In the Model Tree, double-click **Instances** underneath the **Assembly** container and select `plate` as the part to instance.
2. Create a second instance of the plate. Toggle on the option to automatically offset the part instances.
3. From the main menu bar, select **Constraint->Face to Face**. Select the back face of the plate on the right (the second instance) as the face on the movable instance. Select the back face of the plate on the left (the first instance) as the face on the fixed instance. If necessary, flip the arrows so that they point in opposite directions, as shown in *Figure 8*. Set the offset equal to `0.0`.

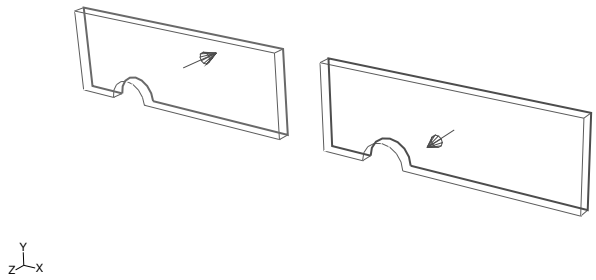


Figure 8: Face-to-face constraint.

4. From the main menu bar, select **Constraint->Parallel Edge**. Select the front top edge of the second plate instance as the edge on the movable instance. Select the front right edge of the first plate instance as the edge on the fixed instance. If necessary, flip the arrows so that they point in the directions shown in *Figure 9*.

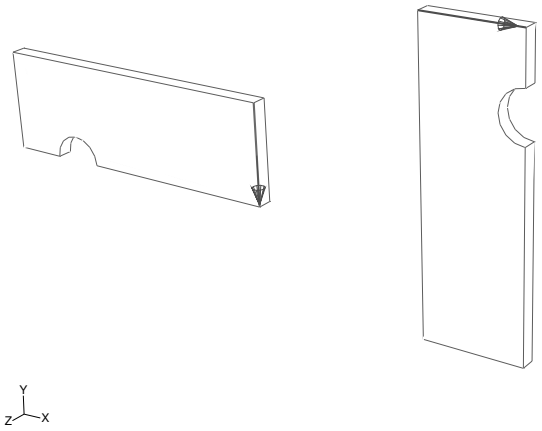


Figure 9: Parallel edge constraint.

5. From the main menu bar, select **Constraint->Coaxial**. Select the cylindrical face of the second plate instance as the face on the movable instance. Select the cylindrical face of the first plate instance as the face on the fixed instance. If necessary, flip the arrows so that they point in the same direction, as shown in *Figure 10*.

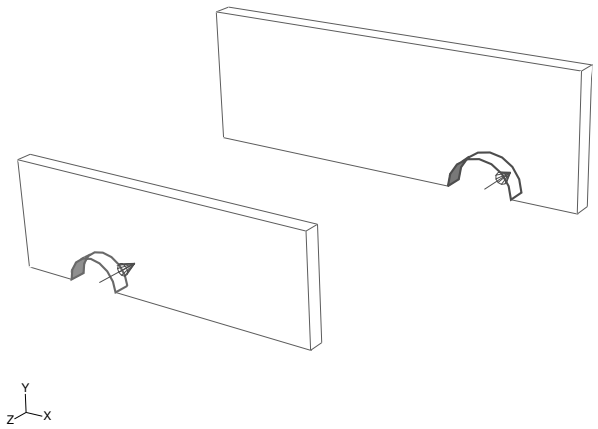


Figure 10: Plate coaxial constraint.

To instance and position the rivet:

1. In the Model Tree, double-click **Instances** underneath the **Assembly** container and select `rivet` as the part to instance.
2. From the main menu bar, select **Constraint->Coaxial**. Select the cylindrical face of the rivet body as the face on the movable instance. Select the cylindrical face of the top plate as the face on the fixed instance. Flip the arrows if necessary so that they point in the directions shown in *Figure 11*.

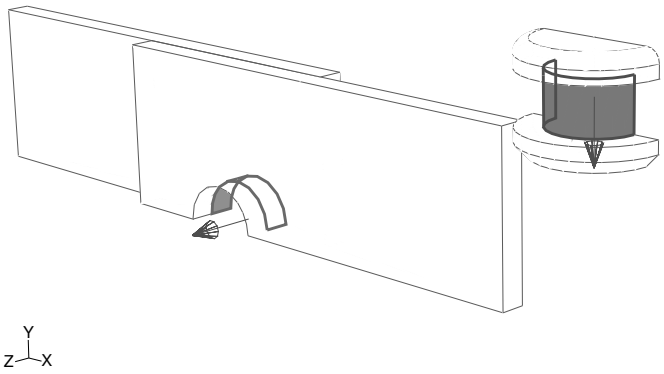


Figure 11: Coaxial constraint.

The final assembly is shown in *Abaqus/Standard 3D example: shearing of a lap joint*.

Geometry sets

At this point it is convenient to create the geometry sets that will be used to specify loads and boundary conditions.

To create geometry sets:

1. Double-click the **Sets** item underneath the **Assembly** container to create the following geometry sets:

- `corner` at the lower left vertex of the bottom plate (*Figure 12*). This set will be used to prevent rigid body motion in the 3-direction.

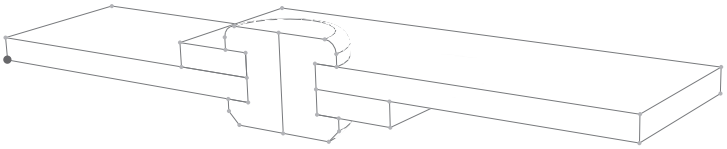


Figure 12: Set `corner`.

- `fix` at the left face of the bottom plate (*Figure 13*). This set will be used to fix the end of the plate.

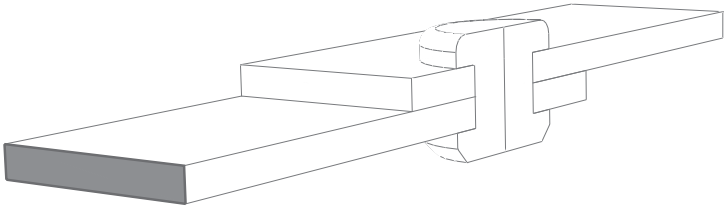


Figure 13: Set `fix`.

- `pull` at the right face of the top plate (*Figure 14*). This set will be used to pull the end of the plate.

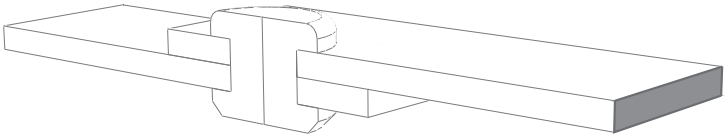


Figure 14: Set `pull`.

- `symm` to include all faces on the symmetry plane (*Figure 15*). This set will be used to impose symmetry conditions.

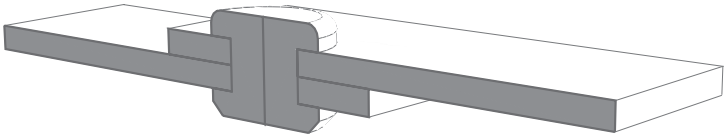


Figure 15: Set `symm`.

Defining steps and output requests

Create a single static, general step after the `Initial` step, and include the effects of geometric nonlinearity. Set the initial time increment to `0.05` and the total time to `1.0`. Accept the default output requests.

Defining contact interactions

Contact will be used to enforce the interactions between the plates and the rivet. The friction coefficient between all parts is assumed to be 0.05.

This problem could use either contact pairs or the general contact algorithm. We will use general contact in this problem to demonstrate the simplicity of the user interface.

Define a contact interaction property named `fric`. In the **Edit Contact Property** dialog box, select **Mechanical->Tangential Behavior**, select **Penalty** as the friction formulation, and specify a friction coefficient of `0.05` in the table. Accept all other defaults.

Create a **General contact (Standard)** interaction named `All` in the `Initial` step. In the **Edit Interaction** dialog box, accept the default selection of **All* with self** for the **Contact Domain** to specify self-contact for the default unnamed, all-inclusive surface defined automatically by Abaqus/Standard. This method is the simplest way to define contact in Abaqus/Standard for an entire model. Select `fric` as the **Global property assignment**, and click **OK**.

Defining boundary conditions

The boundary conditions are defined in the static, general step. The left end of the assembly is fixed while the right end is pulled along the length of the plates (1-direction). A single node is fixed in the vertical (3-) direction to prevent rigid body motion, while the nodes on the symmetry plane are fixed in the direction normal to the plane (2-direction). The boundary conditions are summarized in *Table 1*. Define these conditions in the model.

Table 1: Summary of boundary conditions.

BC Name	Geometry Set	BCs
Fix	fix	U1 = 0.0
Pull	pull	U1 = 2.5
Symmetry	symm	U2 = 0.0
RB	corner	U3 = 0.0

Mesh creation and job definition

The mesh will be created at the part level rather than the assembly level, since all part instances used in this problem are dependent. The dependent instances will inherit the part mesh. Begin by expanding the container for the part named `plate` in the Model Tree, and double-click **Mesh** to switch to the Mesh module.

Mesh the plate with C3D8I elements using a global seed size of `1.2` and the default sweep mesh technique.

Similarly, mesh the rivet with C3D8R elements using a global seed size of `0.5` and the hex-dominated sweep mesh technique. This mesh technique will create wedge-shaped elements (C3D6) about the rivet's axis of symmetry. The meshed assembly appears in *Figure 16*.



Note:

If you are using the Abaqus Learning Edition, these seed sizes will result in a mesh that exceeds the model size limits of the product. For the plate specify a global seed size of 1.75, with a maximum curvature deviation factor of 0.05. For the rivet, specify a global seed size of 1.

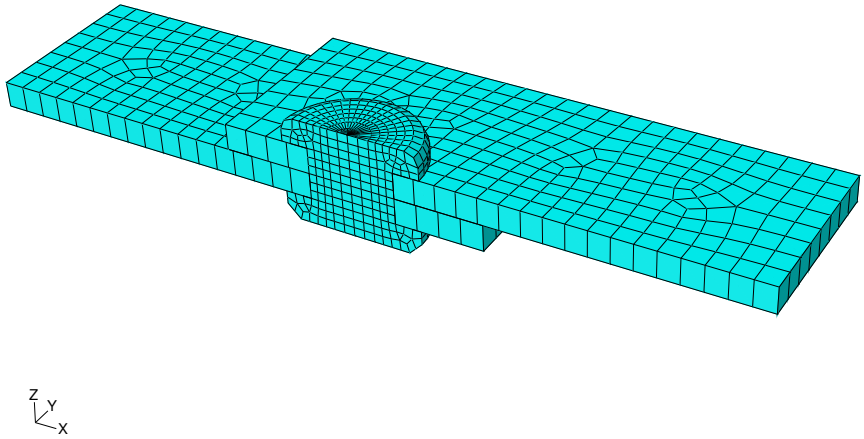


Figure 16: Meshed assembly.

You are now ready to create and run the job. Create a job named `lap_joint`. Save your model to a model database file, and submit the job for analysis. Monitor the solution progress, correct any modeling errors that are detected, and investigate the cause of any warning messages.

Postprocessing

In the Visualization module, examine the deformation of the assembly and other results.

Deformed model shape and contour plots

The basic results of this simulation are the deformation of the joint and the stresses caused by the shearing process. Plot the deformed model shape and the Mises stress, as shown in *Figure 1* and *Figure 2*, respectively.

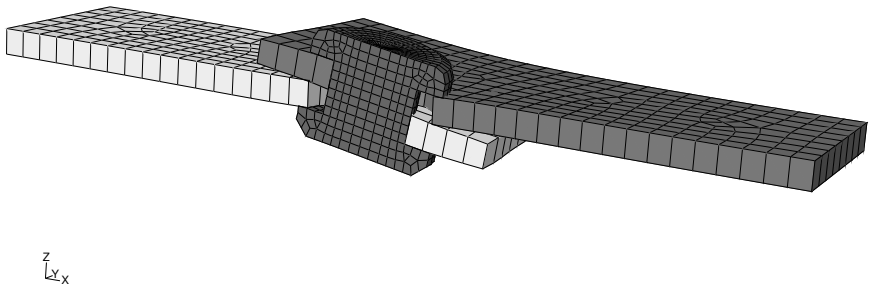


Figure 1: Deformed model shape.

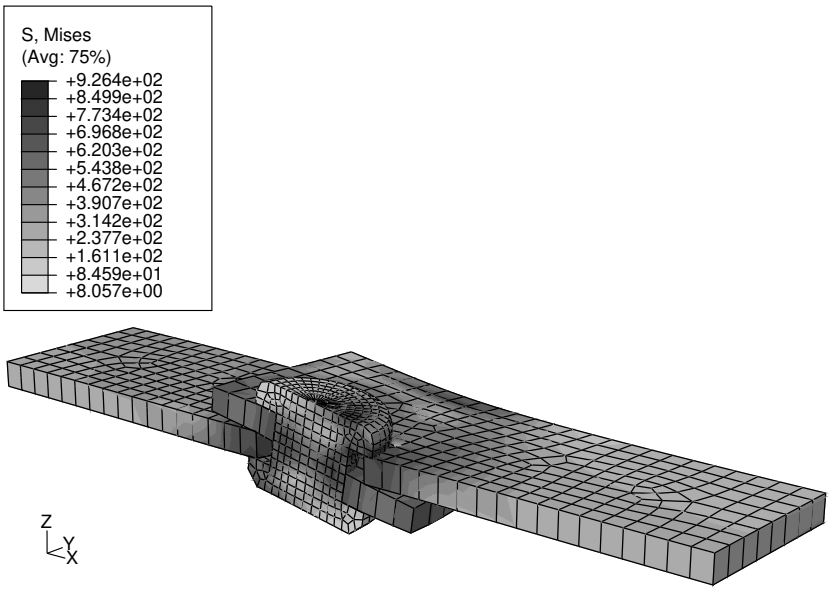


Figure 2: Mises stress.

Contact pressures

You will now plot the contact pressures in the lap joint.

Since it is difficult to see contact pressures when the entire model is displayed, use the **Display Groups** toolbar to display only the top plate in the viewport.

Create a path plot to examine the variation of the contact pressure around the bolt hole of the top plate.

To create a path plot:

1. In the Results Tree, double-click **Paths**. In the **Create Path** dialog box, select **Edge list** as the type and click **Continue**.
2. In the **Edit Edge List Path** dialog box, select the instance corresponding to the top plate and click **Add After**.
3. In the prompt area, select **by shortest distance** as the selection method.
4. In the viewport, select the edge at the left end of the bolt hole as the starting edge of the path and the node at the right end of the bolt hole as the end node of the path, as shown in *Figure 3*.

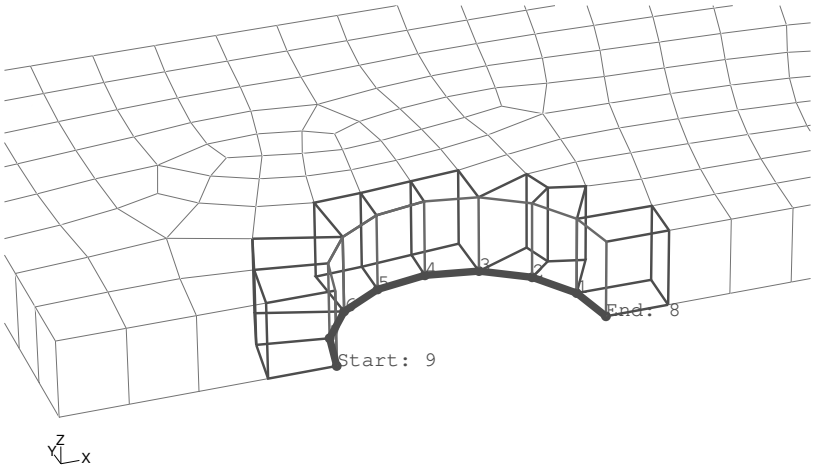


Figure 3: Path definition.

5. Click **Done** in the prompt area to indicate that you have finished making selections for the path. Click **OK** to save the path definition and to close the **Edit Edge List Path** dialog box.
6. In the Results Tree, double-click **XYData**. Select **Path** in the **Create XY Data** dialog box, and click **Continue**.
7. In the **Y Values** frame of the **XY Data from Path** dialog box, click **Step/Frame**. In the **Step/Frame** dialog box, select the last frame of the step. Click **OK** to close the **Step/Frame** dialog box.
8. Make sure that the field output variable is set to **CPRESS**, and click **Plot** to view the path plot. Click **Save As** to save the plot.

   The path plot appears as shown in *Figure 4*.

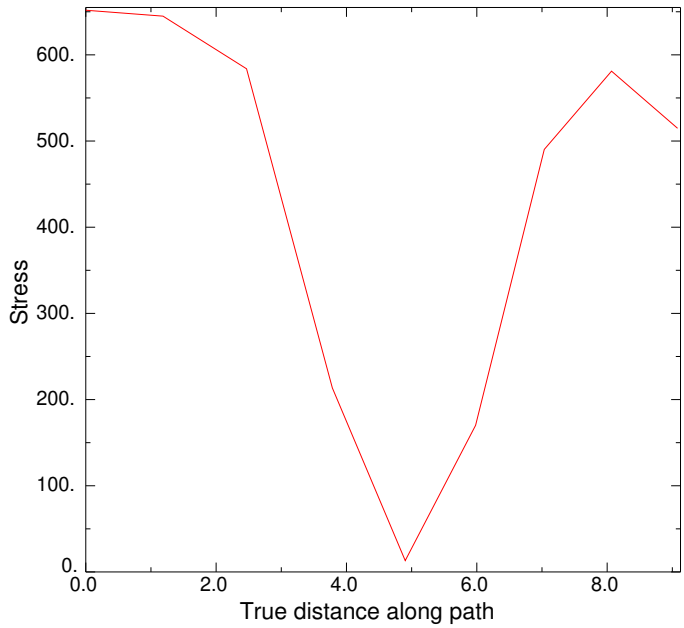


Figure 4: CPRESS distribution around the bolt hole in top plate.

Defining contact in Abaqus/Explicit

Abaqus/Explicit provides two algorithms for modeling contact interactions.
The general (“automatic”) contact algorithm allows very simple definitions of contact with very few restrictions on the types of surfaces involved (see *About General Contact in Abaqus/Explicit*). The contact pair algorithm has more restrictions on the types of surfaces involved and often requires more careful definition of contact; however, it allows for some interaction behaviors that currently are not available with the general contact algorithm (see *About Contact Pairs in Abaqus/Explicit*). General contact interactions typically are defined by specifying self-contact for a default, element-based surface defined automatically by Abaqus/Explicit that includes all bodies in the model. To refine the contact domain, you can include or exclude specific surface pairs. Contact pair interactions are defined by specifying each of the individual surface pairs that can interact with each other.

The contact formulation in Abaqus/Explicit includes the constraint enforcement method, the contact surface weighting, and the sliding formulation.

Constraint enforcement method

For general contact Abaqus/Explicit enforces contact constraints using a penalty contact method, which searches for node-into-face and edge-into-edge penetrations in the current configuration. The penalty stiffness that relates the contact force to the penetration distance is chosen automatically by Abaqus/Explicit so that the effect on the time increment is minimal yet the penetration is not significant.

By default, the contact pair algorithm uses a kinematic contact formulation that achieves precise compliance with the contact conditions using a predictor/corrector method. The increment at first proceeds under the assumption that contact does not occur. If at the end of the increment there is an overclosure, the acceleration is modified to obtain a corrected configuration in which the contact constraints are enforced. The predictor/corrector method used for kinematic contact is discussed in more detail in *Contact Constraint Enforcement Methods in Abaqus/Explicit*; some limitations of this method are discussed in *Common Difficulties Associated with Contact Modeling Using Contact Pairs in Abaqus/Explicit*.

The normal contact constraint for contact pairs can optionally be enforced with the penalty contact method, which can model some types of contact that the kinematic method cannot. For example, the penalty method allows modeling of contact between two rigid surfaces (except when both surfaces are analytical rigid surfaces). When the penalty contact formulation is used, equal and opposite contact forces with magnitudes equal to the penalty stiffness times the penetration distance are applied to the main and secondary nodes at the penetration points. The penalty stiffness is chosen automatically by Abaqus/Explicit and is similar to that used by the general contact algorithm. The penalty stiffness can be overridden for surface-to-surface contact interactions by specifying a penalty scale factor or a “softened” contact relationship.

Contact surface weighting

In the pure main-secondary approach one of the surfaces is the main surface and the other is the secondary surface. As the two bodies come into contact, the penetrations are detected and the contact constraints are applied according to the constraint enforcement method (kinematic or penalty). Pure main-secondary weighting (regardless of the constraint enforcement method) will resist only penetrations of secondary nodes into main facets. Penetrations of main nodes into the secondary surface can go undetected, as shown in *Figure 1*, unless the mesh on the secondary surface is adequately refined.

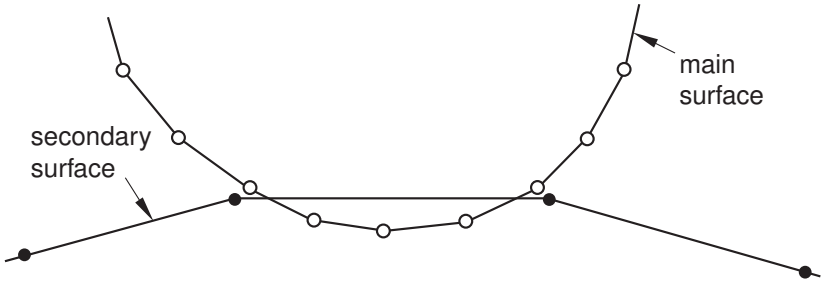


Figure 1: Penetration of main nodes into secondary surface with pure main-secondary contact.

Balanced main-secondary contact simply applies the pure main-secondary approach twice, reversing the surfaces on the second pass. One set of contact constraints is obtained with surface 1 as the secondary, and another set of constraints is obtained with surface 2 as the secondary. The acceleration corrections or forces are obtained by taking a weighted average of the two calculations. For kinematic balanced main-secondary contact a second correction is made to resolve any remaining penetrations, as described in *Contact Formulations for Contact Pairs in Abaqus/Explicit*. The balanced main-secondary contact constraint when kinematic compliance is used is illustrated in *Figure 2*.

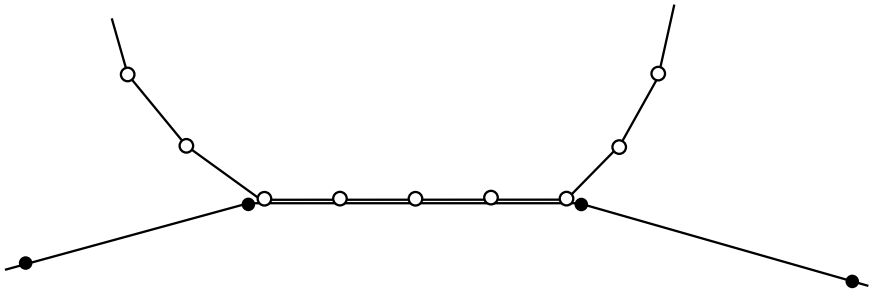


Figure 2: Balanced main-secondary contact constraint with kinematic compliance.

The balanced approach minimizes the penetration of the contacting bodies and, thus, provides more accurate results in most cases.

The general contact algorithm uses balanced main-secondary weighting whenever possible; pure main-secondary weighting is used for general contact interactions involving node-based surfaces, which can act only as pure secondary surfaces. For the contact pair algorithm Abaqus/Explicit will decide which type of weighting to use for a given contact pair based on the nature of the two surfaces involved and the constraint enforcement method used.

Sliding formulation

When you define a surface-to-surface contact interaction, you must decide whether the magnitude of the relative sliding will be small or finite. The default (and only option for general contact interactions) is the more general finite-sliding formulation. The small-sliding formulation is appropriate if the relative motion of the two surfaces is less than a small proportion of the characteristic length of an element face. Using the small-sliding formulation when applicable results in a more efficient analysis.

Modeling considerations in Abaqus/Explicit

We now discuss the following modeling considerations: correct definition of surfaces, overconstraints, mesh refinement, and initial overclosures.

In this section:

- *Correct surface definitions*
- *Overconstraining the model*
- *Mesh refinement*
- *Initial contact overclosure*

Correct surface definitions

Certain rules must be followed when defining surfaces for use with each of the contact algorithms. The general contact algorithm has fewer restrictions on the types of surfaces that can be involved in contact; however, two-dimensional and node-based surfaces can be used only with the contact pair algorithm.

Continuous surfaces

Surfaces used with the general contact algorithm can span multiple unattached bodies. More than two surface facets can share a common edge. In contrast, all surfaces used with the contact pair algorithm must be continuous and simply connected. The continuity requirement has the following implications for what constitutes a valid or invalid surface definition for the contact pair algorithm:

- In two dimensions the surface must be either a simple, nonintersecting curve with two terminal ends or a closed loop. *Figure 1* shows examples of valid and invalid two-dimensional surfaces.

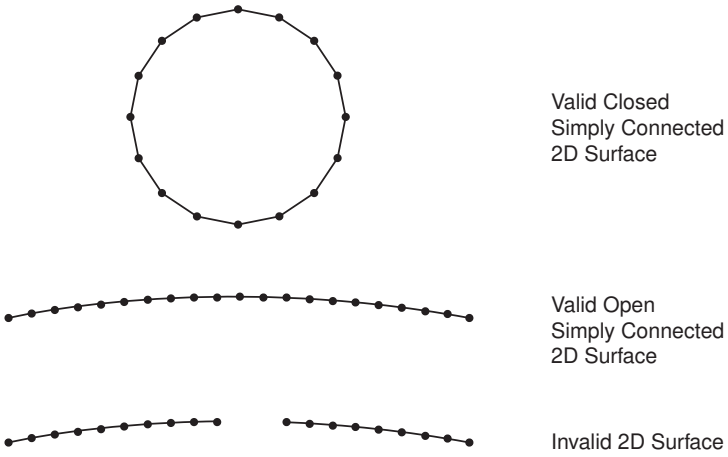


Figure 1: Valid and invalid two-dimensional surfaces for the contact pair algorithm.

- In three dimensions an edge of an element face belonging to a valid surface may be either on the perimeter of the surface or shared by one other face. Two element faces forming a contact surface cannot be joined just at a shared node; they must be joined across a common element edge. An element edge cannot be shared by more than two surface facets. *Figure 2* illustrates valid and invalid three-dimensional surfaces.

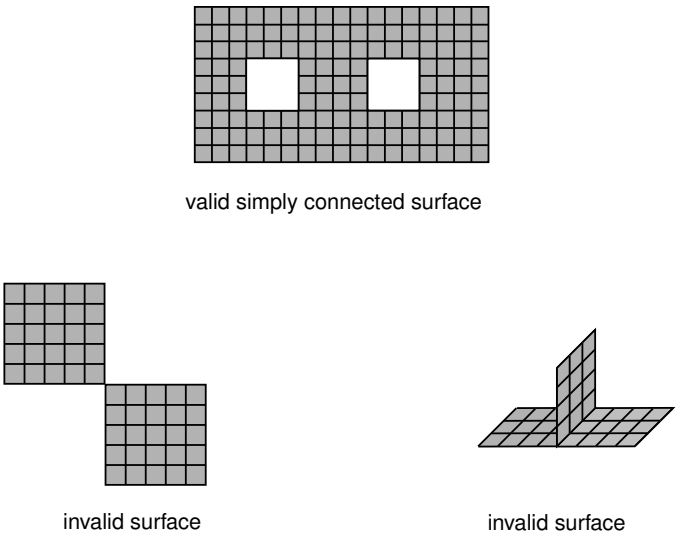


Figure 2: Valid and invalid three-dimensional surfaces for the contact pair algorithm.

- In addition, it is possible to define three-dimensional, double-sided surfaces. In this case both sides of a shell, membrane, or rigid element are included in the same surface definition, as shown in *Figure 3*.

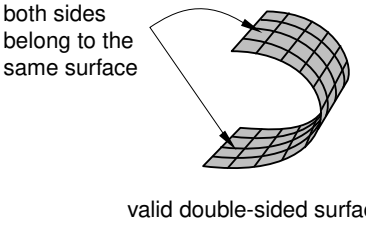


Figure 3: Valid double-sided surface.

Extending surfaces

Abaqus/Explicit does not extend surfaces automatically beyond the perimeter of the surface defined by the user. If a node from one surface is in contact with another surface and it slides along the surface until it reaches an edge, it may "fall off the edge." Such behavior can be particularly troublesome because the node may later reenter from the back side of the surface, thereby violating the kinematic constraint and causing large jumps in acceleration at that node. Consequently, it is good modeling practice to extend surfaces somewhat beyond the regions that will actually contact. In general, we recommend covering each contacting body entirely with surfaces; the additional computational expense is minimal.

Figure 4 shows two simple box-like bodies constructed of brick elements.

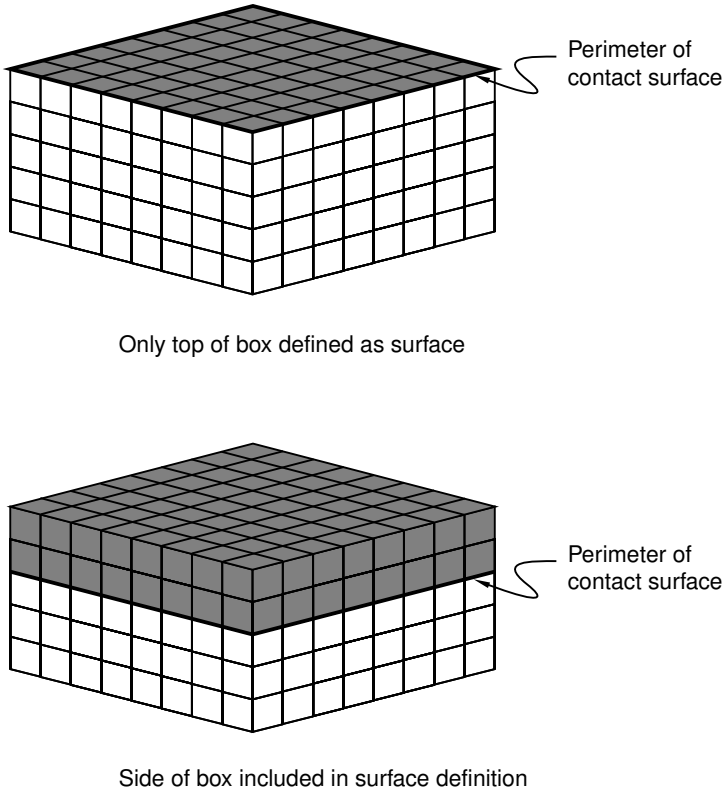


Figure 4: Surface perimeters.

The upper box has a contact surface defined only on the top face of the box. While it is a permissible surface definition in Abaqus/Explicit, the lack of extensions beyond the “raw edge” could be problematic. In the lower box the surface wraps some distance around the side walls, thereby extending beyond the flat, upper surface. If contact is to occur only at the top face of the box, this extended surface definition minimizes contact problems by keeping any contacting nodes from going behind the contact surface.

Mesh seams

Two nodes with the same coordinates (double nodes) can generate a seam or crack in a valid surface that appears to be continuous, as shown in *Figure 5*. A node sliding along the surface can fall through this crack and slide behind the contact surface. A large, nonphysical acceleration correction may be caused once penetration is detected. Surfaces defined in Abaqus/CAE will never have two nodes located at the same coordinates; however, imported meshes can have double nodes. Mesh seams can be detected in the Visualization module by drawing the free edges of the model. Any seams that are not part of the desired perimeter can be double-noded regions.

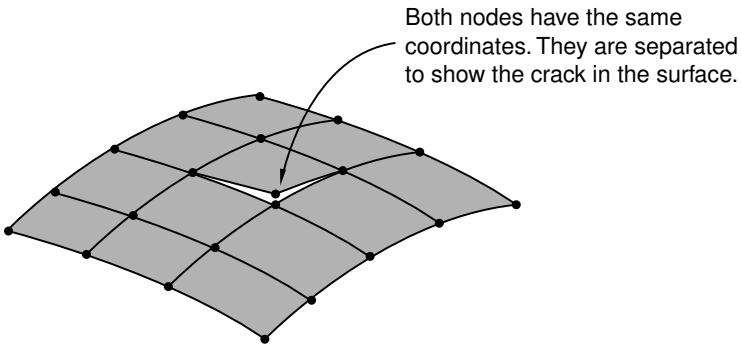


Figure 5: Example of a double-noded mesh.

Complete surface definition

Figure 6 illustrates a two-dimensional model of a simple connection between two parts. The contact definition shown in the figure is not adequate for modeling this connection because the surfaces do not represent a complete description of the geometry of the bodies. At the beginning of the analysis some of the nodes on surface 3 find that they are "behind" surfaces 1 and 2. *Figure 7* shows an adequate surface definition for this connection. The surfaces are continuous and describe the entire geometry of the contacting bodies.

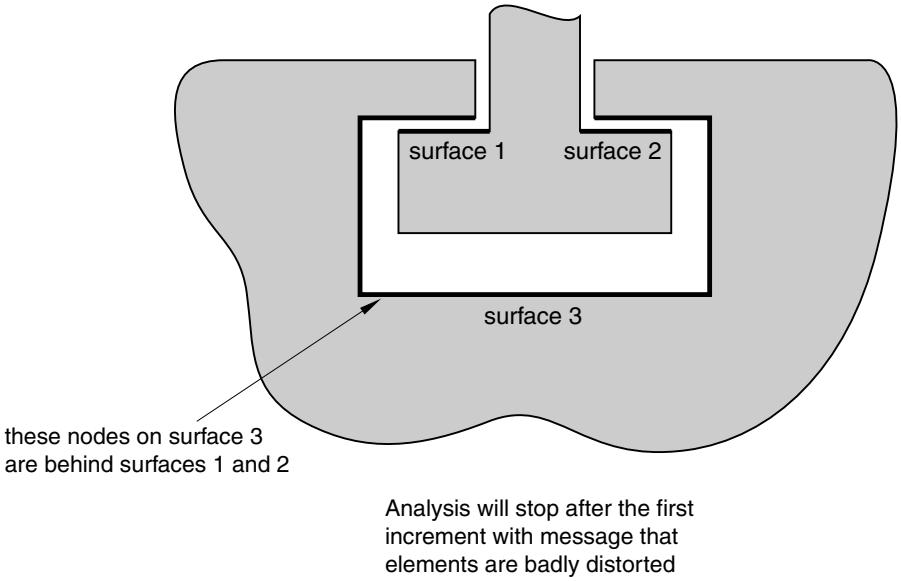


Figure 6: Example of an incorrect surface definition.

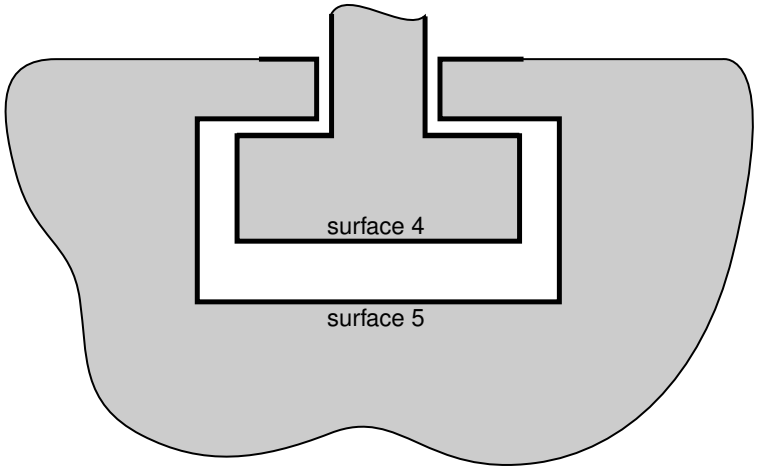


Figure 7: Correct surface definition.

Highly warped surfaces

No special treatment of warped surfaces is required for the general contact algorithm. However, when a surface used with the contact pair algorithm contains highly warped facets, a more expensive tracking approach must be used than the approach required when the surface does not contain highly warped facets. To keep the solution as efficient as possible, Abaqus monitors the warpage of the surfaces and issues a warning if surfaces become too warped; if the normal directions of adjacent facets differ by more than 20°, Abaqus issues a warning message. Once a surface is deemed to be highly warped, Abaqus switches from the more efficient contact search approach to a more accurate search approach to account for the difficulties posed by the highly warped surface.

For purposes of efficiency Abaqus does not check for highly warped surfaces every increment. Rigid surfaces are checked for high warpage only at the start of the step, since rigid surfaces do not change shape during the analysis. Deformable surfaces are checked for high warpage every 20 increments by default; however, some analyses may have surfaces with rapidly increasing warpage, making the default 20 increment frequency check inadequate. The frequency of warping checks can be changed to the desired number of increments. Some analyses in which the surface warping is less than 20° may also require the more accurate contact search approach associated with highly warped surfaces. The angle that defines high warpage can be redefined.

Rigid element discretization

Complex rigid surface geometries can be modeled using rigid elements. Rigid elements in Abaqus/Explicit are not smoothed; they remain faceted exactly as defined by the user. The advantage of unsmoothed surfaces is that the surface defined by the user is exactly the same as the surface used by Abaqus; the disadvantage is that faceted surfaces require much higher mesh refinement to define smooth bodies accurately. In general, using a large number of rigid elements to define a rigid surface does not increase the CPU costs significantly. However, a large number of rigid elements does increase the memory overhead significantly.

The user must ensure that the discretization of any curved geometry on rigid bodies is adequate. If the rigid body discretization is too coarse, contacting nodes on the deformable body may “snag,” leading to erroneous results, as illustrated in *Figure 8*.

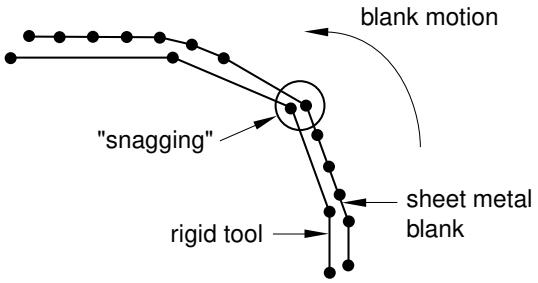


Figure 8: Potential effect of coarse rigid body discretization.

A node that is snagged on a sharp corner may be trapped from further sliding along the rigid surface for some time. Once enough energy is released to slide beyond the sharp corner, the node will snap dynamically before contacting the adjacent facet. Such motions cause noisy solutions. The more refined the rigid surface, the smoother the motion of the contacting secondary nodes. The general contact algorithm includes some numerical rounding of features that prevents snagging of nodes from becoming a concern for discrete rigid surfaces. In addition, penalty enforcement of the contact constraints reduces the tendency for snagging to occur. Analytical rigid surfaces should normally be used with the contact pair algorithm for rigid bodies whose shape is an extruded profile or a surface of revolution.

Overconstraining the model

Just as multiple conflicting boundary conditions should not be defined at a given node, multi-point constraints and contact conditions enforced with the kinematic method generally should not be defined at the same node because they may generate conflicting kinematic constraints. Unless the constraints are entirely orthogonal to one another, the model will be overconstrained; the resulting solution will be quite noisy, as Abaqus/Explicit tries to satisfy the conflicting constraints. Penalty contact constraints and multi-point constraints acting on the same nodes will not generate conflicts because the penalty constraints are not enforced as strictly as the multi-point constraints.

Mesh refinement

For contact as well as all other types of analyses, the solution improves as the mesh is refined. For contact analyses using a pure main-secondary approach, it is especially important that the secondary surface is refined adequately so that the main surface facets do not overly penetrate the secondary surface. The balanced main-secondary approach does not require high mesh refinement on the secondary surface to have adequate contact compliance. Mesh refinement is generally most important with pure main-secondary contact between deformable and rigid bodies; in this case the deformable body is always the pure secondary surface and, thus, must be refined enough to interact with any feature on the rigid body. *Figure 1* shows an example of the penetration that can occur if the discretization of the secondary surface is poor compared to the dimensions of the features on the main surface. If the deformable surface were more refined, the penetrations of the rigid surface would be much less severe.

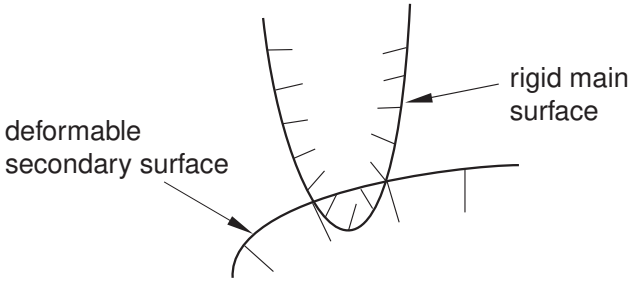


Figure 1: Example of inadequate secondary surface discretization.

Tie constraints

The tie constraint prevents surfaces initially in contact from penetrating, separating, or sliding relative to one another. It is, therefore, an easy means of mesh refinement. Since any gaps that exist between the two surfaces, however small, will result in nodes that are not tied to the opposite boundary, you must adjust the nodes to ensure that the two surfaces are exactly in contact at the start of the analysis.

The tie constraint formulation constrains translational and, optionally, rotational degrees of freedom. When using tied contact with structural elements, you must ensure that any unconstrained rotations will not cause problems.

Initial contact overclosure

Abaqus/Explicit will automatically adjust the undeformed coordinates of nodes on contact surfaces to remove any initial overclosures. When using the balanced main-secondary approach, both surfaces are adjusted; when using the pure main-secondary approach, only the secondary surface is adjusted. Displacements associated with adjusting the surface to remove overclosures do not cause any initial strain or stress for contact defined in the first step of the analysis. When conflicting constraints exist, initial overclosures may not be completely resolved by repositioning nodes. In this case severe mesh distortions can result near the beginning of an analysis when the contact pair algorithm is used. The general contact algorithm stores any unresolved initial penetrations as offsets to avoid large initial accelerations.

In subsequent steps any nodal adjustments to remove initial overclosures cause strains that often cause severe mesh distortions because the entire nodal adjustments occur in a single, very brief increment. This is especially true when the kinematic constraint method is used. For example, if a node is overclosed by 1.0×10^{-3} m and the increment time is 1.0×10^{-7} s, the acceleration applied to the node to correct the overclosure is 2.0×10^{11} m/s^2. Such a large acceleration applied to a single node typically will cause warnings about deformation speed exceeding the wave speed of the material and warnings about severe mesh distortions a few increments later, once the large acceleration has deformed the associated elements significantly. Even a very slight initial overclosure can induce extremely large accelerations for kinematic contact. In general, it is important that in the second step and beyond any new contact surfaces that you define are not overclosed.

Figure 1 shows a common case of initial overclosure of two contact surfaces. All of the nodes on the contact surfaces lie exactly on the same arc of a circle; but since the mesh of the inner surface is finer than that of the outer surface and since the element edges are linear, some nodes on the finer, inner surface initially penetrate the outer surface.

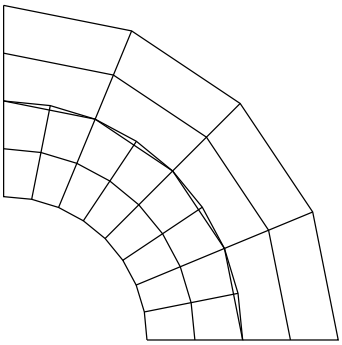


Figure 1: Original overclosure of two contact surfaces.

Assuming that the pure main-secondary approach is used, *Figure 2* shows the initial, strain-free displacements applied to the secondary-surface nodes by Abaqus/Explicit. In the absence of external loads this geometry is stress free. If the default, balanced main-secondary approach is used, a different initial set of displacements is obtained, and the resulting mesh is not entirely stress free.

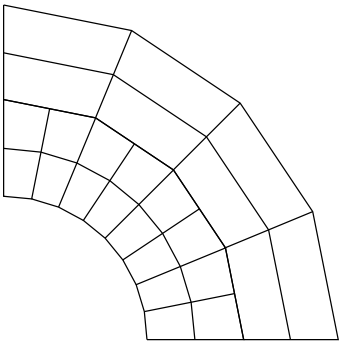


Figure 2: Corrected contact surfaces.

Abaqus/Explicit example: circuit board drop test

This example examines the results of a circuit board drop test.

In this section:

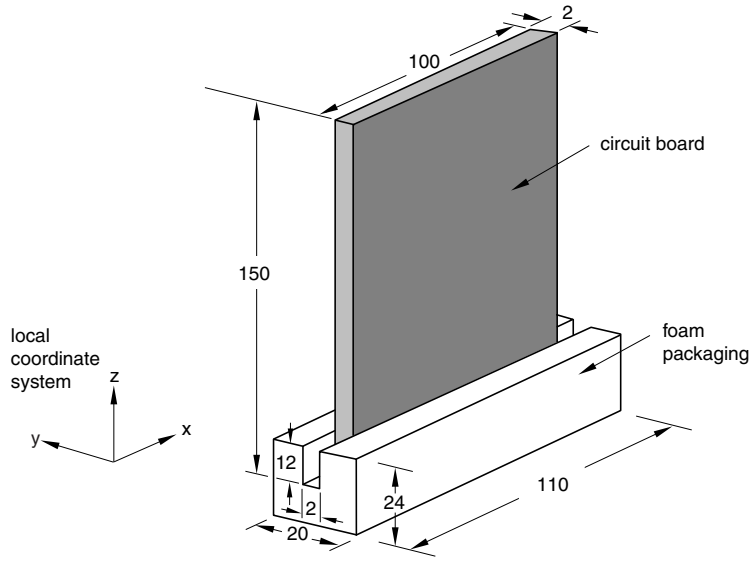
- *Circuit board drop test analysis*
- *Preprocessing—creating the model with Abaqus/CAE*
- *Postprocessing*
- *Rerunning the analysis with output filtering*

Circuit board drop test analysis

In this example you will investigate the behavior of a circuit board in protective crushable foam packaging dropped at an angle onto a rigid surface.

Your goal is to assess whether the foam packaging is adequate to prevent circuit board damage when the board is dropped from a height of 1 meter. You will use the general contact capability in Abaqus/Explicit to model the interactions between the different components.

Figure 1 shows the dimensions of the circuit board and foam packaging in millimeters and the material properties.



Material properties

Circuit board material (plastic):

$E = 45 \times 10^9$ Pa

$\nu = 0.3$

$\rho = 500$ kg/m^3

Foam packaging material is crushable foam:

$E = 3 \times 10^6$ Pa

$\nu = 0.0$

$\rho = 100$ kg/m^3

(Foam plasticity data are given in the text.)

Figure 1: Dimensions in millimeters and material properties.

Preprocessing—creating the model with Abaqus/CAE

Create the model for this simulation with Abaqus/CAE. Abaqus provides scripts that replicate the complete analysis model for this problem.
Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Circuit board drop test*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Circuit board drop test**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

In this section:

- *Defining the model geometry*
- *Defining the material and section properties*
- *Creating the assembly*
- *Defining the step and requesting output*
- *Defining contact*
- *Defining tie constraints*
- *Assigning mass properties to the chips*
- *Specifying loads and boundary conditions*
- *Meshing the model and defining a job*

Defining the model geometry

You will create three parts representing the packaging, the circuit board, and the floor.
The chips will be represented using discrete point masses. You will also create a number of datum points to aid in positioning the part instances and the point masses.

Define the packaging geometry

1. The packaging is a three-dimensional solid structure. Create a three-dimensional, deformable part with an extruded solid base feature to represent the packaging; name the part `Packaging`. Use an approximate part size of `0.1`, and sketch a 0.02 m × 0.024 m rectangle as the profile. Specify `0.11` m as the extrusion depth.
2. From the main menu bar, select **Shape->Cut->Extrude** to create the cut in the packaging in which the circuit board will rest.
   a. Select the left end of the packaging as the plane for the extruded cut. Select a vertical line on the packaging profile to be vertical and on the right in the sketching plane.
   b. In the Sketcher, use the vertical construction line tool  to create a vertical construction line through the center of the packaging. Apply a fixed constraint to the construction line.
   c. Sketch the profile of the cut shown in *Figure 1*. Use a **Symmetry** constraint to center the cut about the construction line and edit the dimensions of the cut profile so that it is 0.002 m wide and extends a distance of 0.012 m into the packaging.

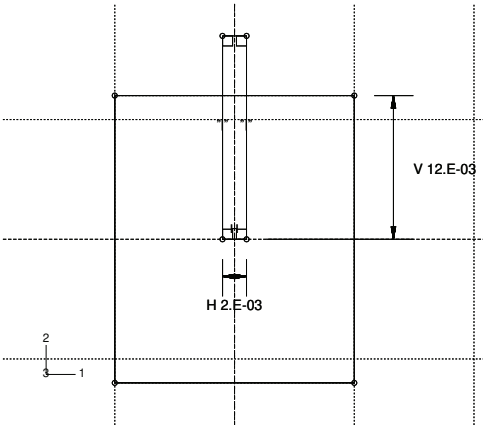


Figure 1: Profile of cut in packaging (with grid spacing doubled).

   d. In the **Edit Cut Extrusion** dialog box that appears upon completion of the sketch, select **Through All** as the end condition and select the arrow direction representing a cut into the packaging.
3. Create a datum point centered on the bottom face of the cut, as shown in *Figure 2*. This point will be used to position the board relative to the packaging.

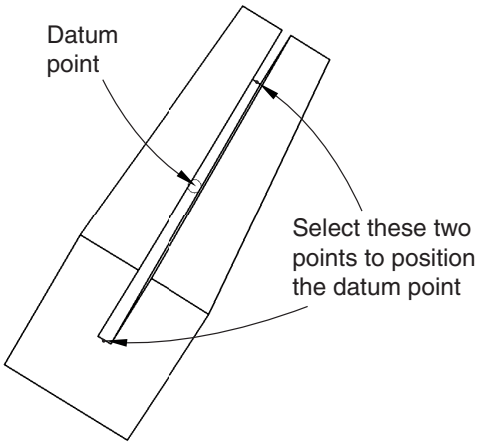


Figure 2: Datum point at center of cut in packaging.

a. From the main menu bar, select **Tools->Datum**.

The **Create Datum** dialog box appears.

b. Accept the default selection of **Point** as the datum type and select **Midway between 2 points** as the method.

c. Select the two points centered on the bottom face at either end of the cut to be the two points between which the datum point will be created.

Abaqus/CAE creates the datum point shown in *Figure 2*.

Define the circuit board geometry

1. The circuit board can be modeled as a thin, flat plate with chips attached to it. Create a three-dimensional, deformable planar shell to represent the circuit board; name the part `Board`. Use an approximate part size of `0.5`, and sketch a 0.100 m × 0.150 m rectangle for the profile.
2. Create the three datum points shown in *Figure 3*. These points will be used to position the chips on the board.

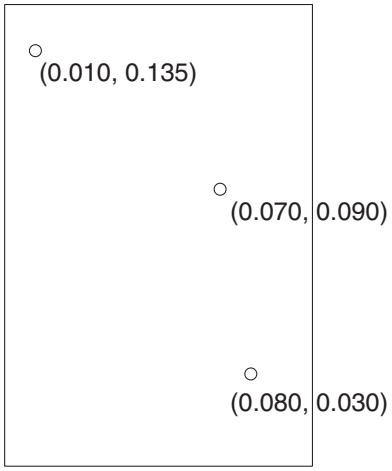


Figure 3: Datum points used to position the chips relative to the board. Numbers in parentheses are (*x*, *y*) coordinates in meters based on a local origin at the bottom left corner of the circuit board.

a. From the main menu bar, select **Tools->Datum**.

The **Create Datum** dialog box appears.

b. Accept the default selection of **Point** as the datum type, and select **Offset from point** as the method.

c. Select the bottom left corner of the board as the point from which to offset, and enter the coordinates for one of the points shown in *Figure 3*.

d. Repeat Step c to create the other two datum points.

Define the floor

1. The surface that the circuit board will impact is effectively rigid. Create a three-dimensional, discrete rigid planar shell to represent the floor; name the part `Floor`. Use an approximate part size of `0.5`. The rigid surface should be large enough to keep any deformable bodies from falling off the edges.
2. Sketch a 0.2 m × 0.2 m square as the profile. To simplify the positioning of parts in the assembly of the model, ensure that the center of the surface corresponds to point (0, 0) in the sketcher. This also corresponds to the origin of the global coordinate system.
3. Assign a reference point at the center of the part.

Defining the material and section properties

You can define the material and section properties.

The circuit board is assumed to be made of a PCB elastic material with a Young's modulus of 45×10^9 Pa and a Poisson's ratio of 0.3. The mass density of the board is 500 kg/m^3.

Define a material named `PCB` with these properties.

The foam packaging material will be modeled using the crushable foam plasticity model. The elastic properties of the packaging include a Young's modulus of 3×10^6 Pa and a Poisson's ratio of 0.0. The material density of the packaging is 100 kg/m^3. Define a material named `Foam` with these properties; do not close the material editor.

The yield surface of a crushable foam in the *p–q* (pressure stress-Mises equivalent stress) plane is illustrated in *Figure 1*.

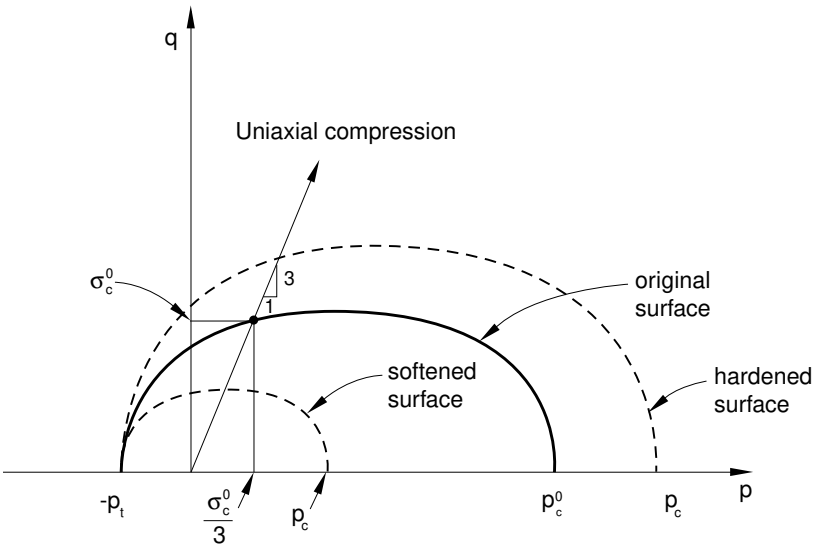


Figure 1: Crushable foam model: yield surface in the *p–q* plane.

The initial yield behavior is governed by the ratio of initial yield stress in uniaxial compression to initial yield stress in hydrostatic compression, σ_c^0/p_c^0, and the ratio of yield stress in hydrostatic tension to initial yield stress in hydrostatic compression, p_t/p_c^0. In this problem we have chosen the first data item to be 1.1 and the second data item (which is given as a positive value) to be 0.1.

Hardening effects are also included in the material model definition. *Table 1* summarizes the yield stress–plastic strain data.

Table 1: Yield stress–plastic strain data for the crushable foam model.

Yield stress in uniaxial compression (Pa)	Plastic strain
0.22000E6	0.0
0.24651E6	0.1
0.27294E6	0.2
0.29902E6	0.3
0.32455E6	0.4

Yield stress in uniaxial compression (Pa)	Plastic strain
0.34935E6	0.5
0.37326E6	0.6
0.39617E6	0.7
0.41801E6	0.8
0.43872E6	0.9
0.45827E6	1.0
0.49384E6	1.2
0.52484E6	1.4
0.55153E6	1.6
0.57431E6	1.8
0.59359E6	2.0
0.62936E6	2.5
0.65199E6	3.0
0.68334E6	5.0
0.68833E6	10.0

The crushable foam hardening model follows the curve shown in *Figure 2*.

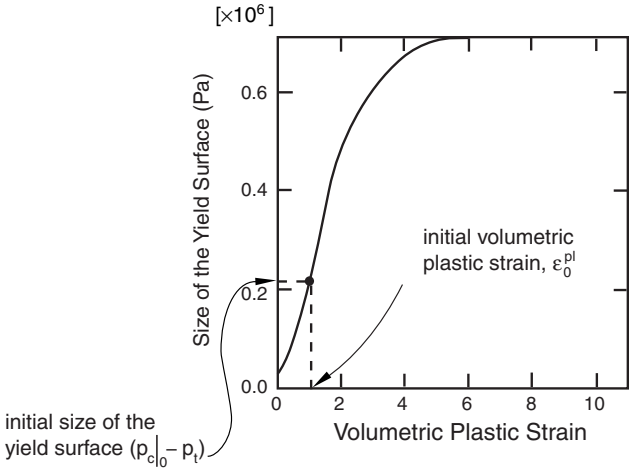


Figure 2: Foam hardening material data.

For your convenience, the hardening data are available in a text file named `drop-test-foam.txt`. Enter the following command at the operating system prompt to use the Abaqus**fetch** utility to copy the file to your local directory:

```
abaqus fetch job=drop*.txt
```

In the material editor, select **Mechanical->Plasticity->Crushable Foam**. Enter the yield stress ratios given above. Click **Suboptions**, and select **Foam Hardening**. Select the first cell in the suboption editor, and click mouse button 3. From the menu that appears, select **Read from File**. Select the file named `drop-test-foam.txt` to read the hardening data shown in *Table 1*.

Define a homogeneous shell section named `BoardSection` that refers to the material `PCB`. Specify a thickness of `0.002` m, and assign this section definition to the part `Board`. Define a homogeneous solid section named `FoamSection` that refers to the material `Foam`. Assign this section definition to the part `Packaging`.

For the circuit board it is most meaningful to output stress results in the longitudinal and lateral directions, aligned with the edges of the board. Therefore, you need to specify a local material orientation for the circuit board mesh.

1. Double-click **Board** underneath the **Parts** container in the Model Tree.
2. To define a datum coordinate system for the orientation:
   a. From the main menu bar, select **Tools->Datum**.
   b. Select **CSYS** as the type and **2 lines** as the method.
   c. In the **Create Datum CSYS** dialog box that appears, select a rectangular coordinate system and click **Continue**.
   d. In the viewport, select the bottom horizontal edge of the board to be the local *x*-axis and the right vertical edge of the board to lie in the *X*–*Y* plane.

      The datum coordinate system appears in yellow in the viewport.
3. From the main menu bar of the Property module, select **Assign->Material Orientation**. In the viewport select the circuit board. Select the datum coordinate system as the coordinate system. In the material orientation editor select **Axis 3** as the shell surface normal and **None** as the additional rotation about that axis.

   The material orientation appears on the board in the viewport.

Creating the assembly

Next, you create the assembly.

Begin by creating a dependent instance of the floor (double-click **Instances** underneath the **Assembly** container).

The circuit board will be dropped at an angle; the final model assembly is shown in *Figure 1*.

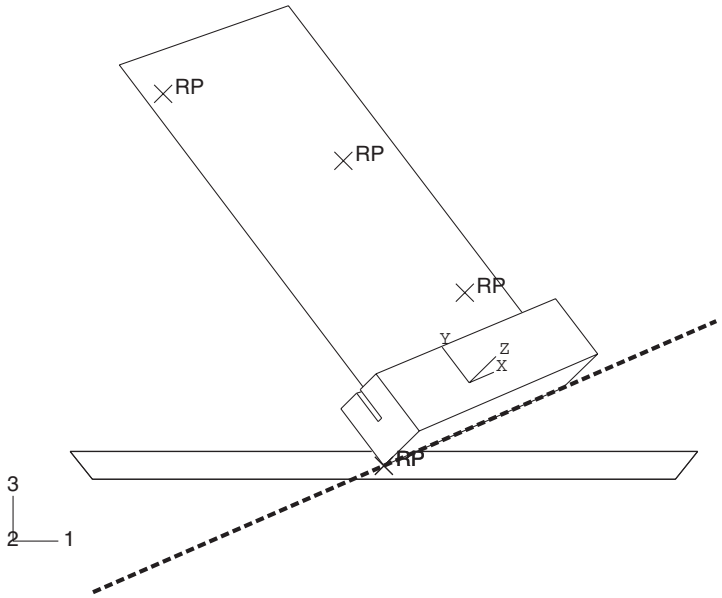


Figure 1: Complete circuit board assembly.

You will use the positioning tools in the Assembly module to position the packaging first; then you will position the board relative to the packaging. Finally, you will create a reference point at each datum point location of the board to represent the chips.

Position the packaging

1. From the main menu bar of the Assembly module, select **Tools->Datum** to create additional datum points that will help you position the packaging.
   a. Select **Point** as the type, and select **Enter coordinates** as the method.
   b. Create two datum points at `(0, 0, 0)` and `(0.5, 0.707, 0.25)`.
   c. Click the auto-fit tool to see both points in the viewport.
2. In the **Create Datum** dialog box, select **Axis** as the type and **2 points** as the method. Create a datum axis defined by the two datum points created in the previous step, selecting the point at (0.5, 0.707, 0.25) as the first point in the datum axis definition.

   Tip: Use the **Selection** toolbar to restrict your selection to **Datums**.

3. Instance the packaging.
4. Constrain the packaging so that the bottom edge aligns with the datum axis.
   a. From the main menu bar, select **Constraint->Edge to Edge**.
   b. Select the edge of the packaging shown in *Figure 2* as a straight edge of the movable instance.

Tip: To obtain a better view of the model, select **View->Specify** from the main menu bar and select **Viewpoint** as the method; enter `(-1, -1, 1)` for the viewpoint vector and `(0, 0, 1)` for the up vector.

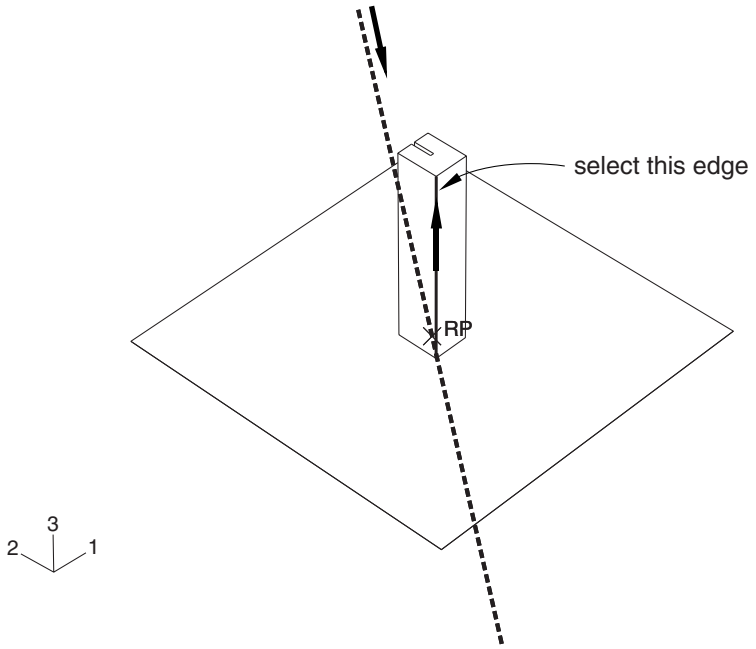


Figure 2: Select a straight edge on the movable instance.

c. Select the datum axis as the fixed instance.

d. If necessary, click **Flip** in the prompt area to reverse the direction of the arrow on the packaging; click **OK** when the arrows point in opposite directions as shown in *Figure 2*.

Tip: You may need to zoom out and rotate the model to see the arrow on the datum axis. The direction of this arrow depends on how you defined the axis initially; if the arrow on your axis points in the reverse direction of the one shown in the figure, the arrow on your packaging should also be opposite to the figure.

Abaqus/CAE positions the packaging as shown in *Figure 3*.

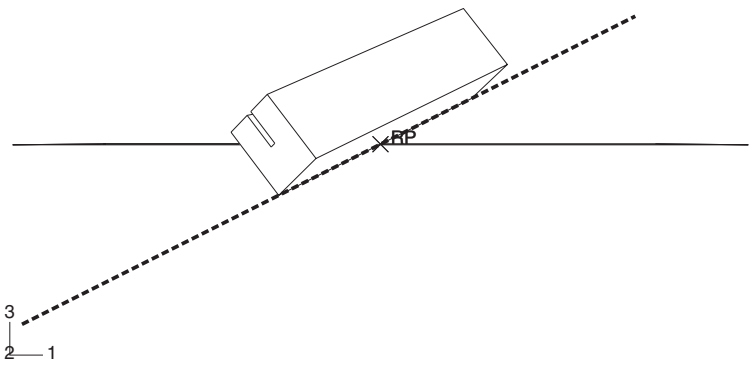


Figure 3: Position 1: Constrain the bottom edge of the packaging to lie along the datum axis.

Note: Abaqus/CAE stores position constraints as features of the assembly; if you make a mistake while positioning the assembly, you can delete the position constraints. Simply click mouse button 3 on the constraint you wish to delete in the list of **Position Constraints** items found underneath the **Assembly** container in the Model Tree, and select **Delete** from the menu that appears.

5. Create a third datum point at `(-0.5, 0.707, -0.5)`, and click the auto-fit tool again.
6. In the **Create Datum** dialog box, select **Plane** as the type and **Line and point** as the method. Create a datum plane defined by the datum axis created earlier and the datum point created in the previous step.
7. Constrain the packaging so that the bottom face lies on the datum plane.
   a. From the main menu bar, select **Constraint->Face to Face**.
   b. Select the face of the packaging shown in *Figure 4* as a face of the movable instance.

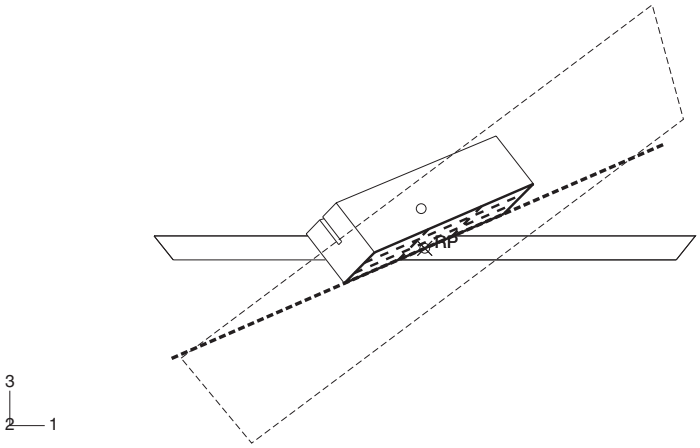


Figure 4: Select a face on the movable instance.

   c. Select the datum plane as the fixed instance.

d. If necessary, click **Flip** in the prompt area; click **OK** when both arrows point in the same direction.

e. Accept the default distance of `0.0` from the fixed plane.

8. Finally, constrain the packaging to contact the floor at its center.

a. From the main menu bar, select **Constraint->Coincident Point**.

b. Select the lowest vertex of the packaging as a point on the movable instance, and select the reference point on the floor as a point on the fixed instance.

Abaqus/CAE positions the packaging as shown in *Figure 5*.

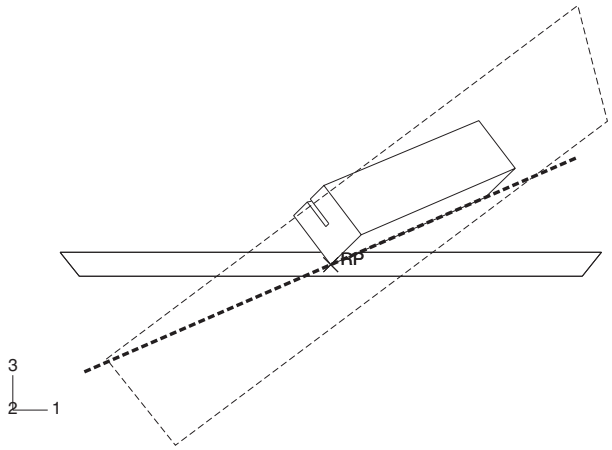


Figure 5: Final position of the packaging relative to the floor.

9. Now, translate the floor slightly downward to ensure that there is no initial overclosure between the packaging and the floor.

a. Convert the relative position constraints to absolute constraints to avoid conflicts. From the main menu bar, select **Instance->Convert Constraints**. Select the packaging in the viewport, and click **Done** in the prompt area.

b. From the main menu bar, select **Instance->Translate**.

c. Select the floor in the viewport.

d. Enter `(0.0, 0.0, 0.0)` as the start point for the translation vector and `(0.0, 0.0, -0.0001)` as the end point for the translation vector.

e. Click **OK** to accept the new position.

Position the circuit board

1. Instance the circuit board. In the **Create Instance** dialog box, toggle on **Auto-offset from other instances**.

2. From the main menu bar, select **Constraint->Parallel Face**. Select the face of the board as a face on the movable instance; select a face on the long side of the packaging as a face on the fixed instance. If necessary, click **Flip** in the prompt area to ensure that the arrows on both faces point in the directions shown in *Figure 6*; click **OK** to complete the constraint.

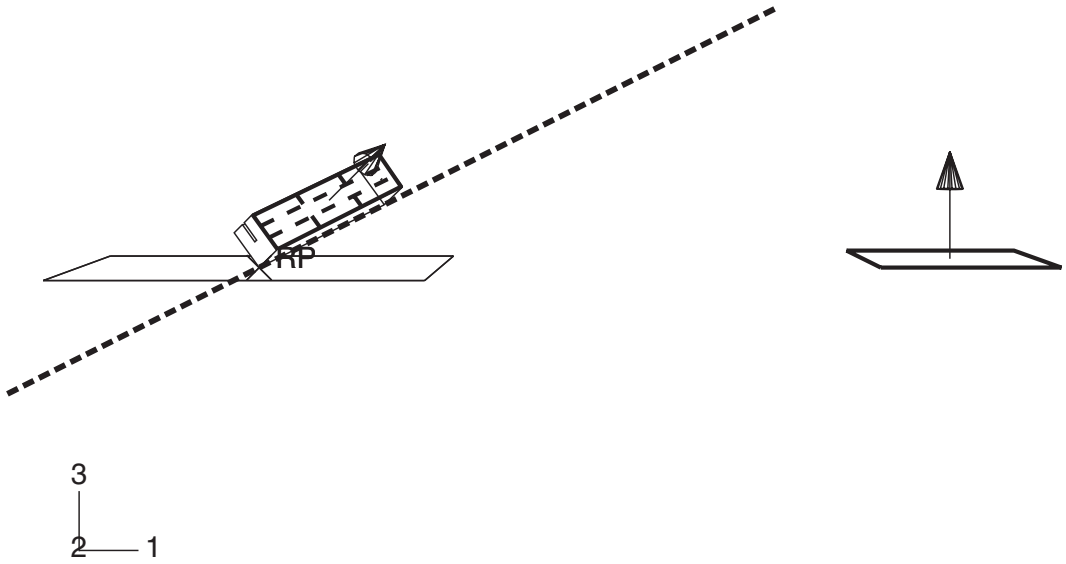


Figure 6: Parallel face constraint for the circuit board.

3. From the main menu bar, select **Constraint->Parallel Edge**. Select the top edge of the board as an edge on the movable instance. Select an edge along the length of the packaging as an edge on the fixed instance. If necessary, click **Flip** in the prompt area to ensure that the arrows point as shown in *Figure 7*; click **OK** to complete the constraint.

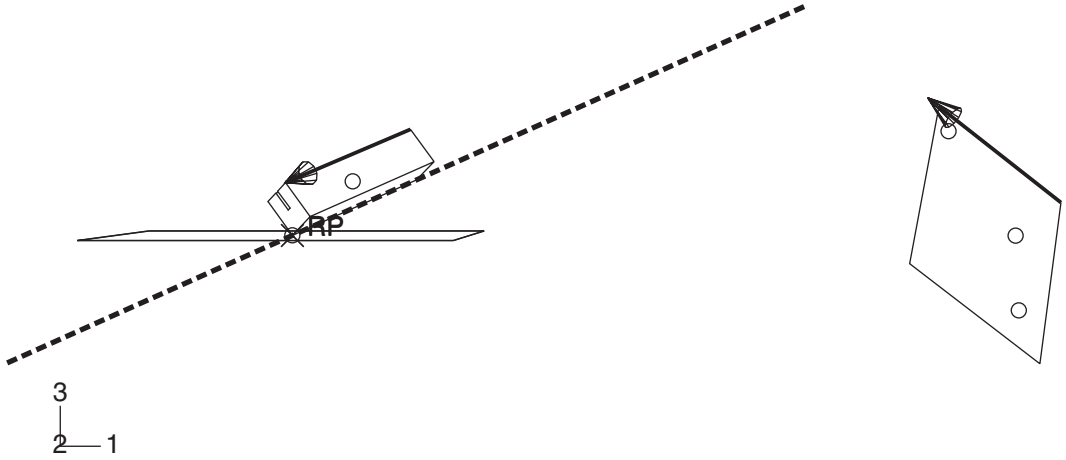


Figure 7: Parallel edge constraint for the circuit board.

4. From the main menu bar, select **Constraint->Coincident Point**. Select the midpoint of the bottom of the board as a point on the movable instance. Select the datum point at the center of the cut in the packaging as a point on the fixed instance.



Tip: Use the **Selection** toolbar to restrict your selection to **Datums**.

Figure 8 shows the final position of the circuit board. The circuit board and the slot in the packaging are the same thickness (2 mm) so there is a snug fit between the two bodies.

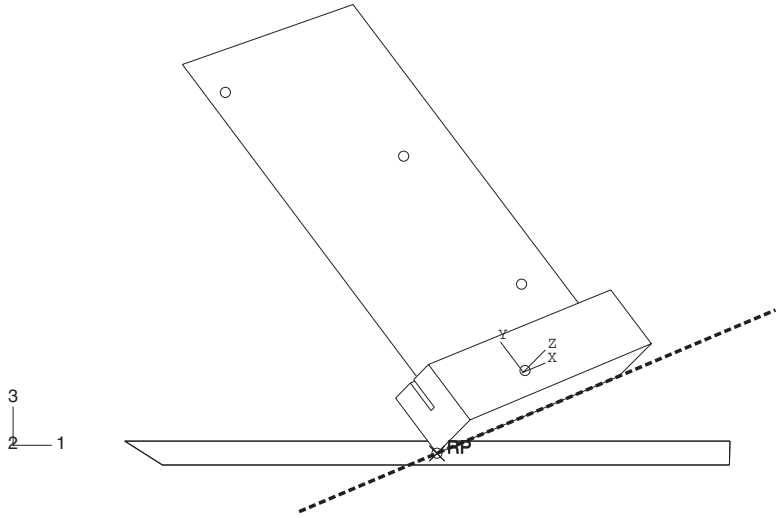


Figure 8: Final position of the circuit board.

Create the chips

1. Create a reference point at each of the three datum point locations on the board to represent each chip. Each of these reference points will later be assigned mass properties. To create a reference point, select **Tools->Reference Point** from the main menu bar of the Assembly module.

   Once you have created the reference points, the assembly is complete.

2. Before continuing, create the following geometry sets that you will use to specify output requests and mass properties:

   - `TopChip` for the reference point of the top chip
   - `MidChip` for the reference point of the middle chip
   - `BotChip` for the reference point of the bottom chip
   - `BotBoard` for the bottom edge of the board

Defining the step and requesting output

You can define an explicit step and request output.

Create a single dynamic, explicit step named `Drop`; set the time period to `0.02` s. Accept the default history and field output requests. In addition, request vertical displacement (**U3**), velocity (**V3**), and acceleration (**A3**) history output every 7×10^{-5} s for each of the three chips.



Tip: Define the history output request for the first chip; using the **History Output Requests Manager**, copy the request and edit the domain to define the requests for the other chips.

Request history output every 7×10^{-5} s for the logarithmic strain components (LE11, LE22, and LE12) and the principal logarithmic strains (LEP) at the top face (section point 5) of the set `BotBoard`.

Defining contact

Either contact algorithm in Abaqus/Explicit could be used for this problem. However, the definition of contact using the contact pair algorithm would be more cumbersome since, unlike general contact, the surfaces involved in contact pairs cannot span more than one body. We use the general contact algorithm in this example to demonstrate the simplicity of the contact definition for more complex geometries.

Define a contact interaction property named `Fric`. In the **Edit Contact Property** dialog box, select **Mechanical->Tangential Behavior**, select **Penalty** as the friction formulation, and specify a friction coefficient of `0.3` in the table. Accept all other defaults.

Create a **General contact (Explicit)** interaction named `All` in the `Drop` step. In the **Edit Interaction** dialog box, accept the default selection of **All* with self** for the **Contact Domain** to specify self-contact for the default unnamed, all-inclusive surface defined automatically by Abaqus/Explicit. This method is the simplest way to define contact in Abaqus/Explicit for an entire model. Select `Fric` as the **Global property assignment**, and click **OK**.

Defining tie constraints

You will use tie constraints to attach the chips to the board.

Begin by defining a surface named `Board` for the circuit board. Select **Both sides** in the prompt area to specify that the surface is double-sided. In the Model Tree, double-click the **Constraints** container; define a tie constraint named `TopChip`. Select `Board` as the main surface and `TopChip` as the secondary node region. Toggle off **Tie rotational DOFs if applicable** in the **Edit Constraint** dialog box since only the effects of the chip masses are of interest, and click **OK**. Yellow circles appear on the model to represent the constraint. Similarly, create tie constraints named `MidChip` and `BotChip` for the middle and bottom chips.

Assigning mass properties to the chips

You will assign a point mass to each chip.

To do this, expand **Engineering Features** underneath the **Assembly** container in the Model Tree. In the list that appears, double-click **Inertias**. In the **Create Inertia** dialog box, enter the name `MassTopChip` and click **Continue**. Select the set **TopChip**, and assign it a mass of `0.005` kg. Repeat this procedure for the two remaining chips.

Specifying loads and boundary conditions

Constrain the reference point on the floor in all directions; for example, you could prescribe an **ENCASTRE** boundary condition.

Two methods could be used to simulate the circuit board being dropped from a height of 1 m. You could model the circuit board and foam at a height of 1 m above the floor and allow Abaqus/Explicit to calculate the motion under the influence of gravity; however, this method is impractical because of the large number of increments required to complete the “free-fall” part of the simulation. The more efficient method is to model the circuit board and packaging in an initial position very close to the surface of the floor (as you have done in this problem) and specify an initial velocity of 4.43 m/s to simulate the 1 m drop. Create a predefined field in the initial step to specify an initial velocity of **V3** = `-4.43` m/s for the board, chips, and packaging.

Meshing the model and defining a job

Next, you mesh the model and define the analysis job.

Seed the circuit board with 10 elements along its length and height. Seed the edges of the packaging as shown in *Figure 1*.

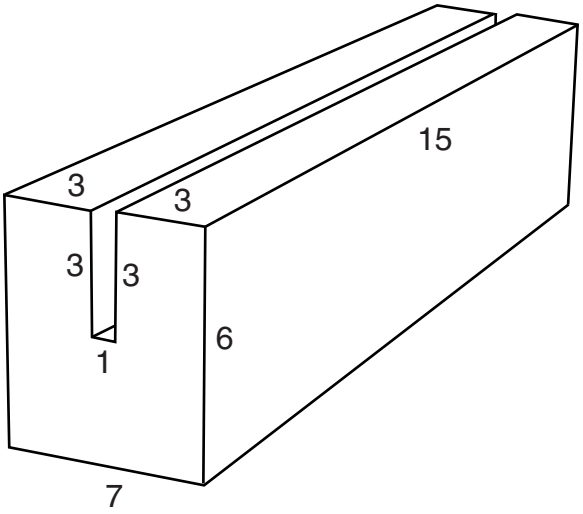


Figure 1: Edge seeds for the packaging mesh.

The mesh for the packaging is too coarse near the impacting corner to provide highly accurate results; however, it is adequate for a low-cost preliminary study. Using the swept mesh technique (with the medial axis algorithm), mesh the packaging with C3D8R elements and the board with S4R elements from the Abaqus/Explicit library. Use enhanced hourglass control for the packaging mesh to control hourglassing effects. Specify a global seed of 1.0 for the floor, and mesh it with one Abaqus/Explicit R3D4 element.



Note:

The suggested mesh density exceeds the model size limits of the Abaqus Learning Edition. Specify 12 elements along the length of the foam packaging if using this product.

Create a job named `Circuit`, and give it the following description: `Circuit board drop test`. You should use double precision for this analysis to minimize the noise in the solution. In the **Precision** tabbed page of the job editor, select **Double-analysis only** as the Abaqus/Explicit precision. Save your model to a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate the cause of any warning messages.

Postprocessing

Enter the Visualization module, and open the output database file created by this job (`Circuit.odb`).

Checking material directions

The material directions obtained from the orientation definition can be checked in the Visualization module.

To plot the material orientation:

1. First, change the view to a more convenient setting. If it is not visible, display the Views toolbar by selecting **View->Toolbars->Views** from the main menu bar. In the Views toolbar, select the X–Z setting.
2. From the main menu bar, select **Plot->Material Orientations->On Deformed Shape**.

   The orientations of the material directions for the circuit board at the end of the simulation are shown. The material directions are drawn in different colors. The material 1-direction is blue, the material 2-direction is yellow, and the 3-direction, if it is present, is red.
3. To view the initial material orientation, select **Result->Step/Frame**. In the **Step/Frame** dialog box that appears, select `Increment 0`. Click **Apply**.

   Abaqus displays the initial material directions.
4. To restore the display to the results at the end of the analysis, select the last increment available in the **Step/Frame** dialog box; and click **OK**.

Animation of results

You will create a time-history animation of the deformation to help you visualize the motion and deformation of the circuit board and foam packaging during impact.

To create a time-history animation:

1. Plot the deformed model shape at the end of the analysis.
2. From the main menu bar, select **Animate->Time History**.

   The animation of the deformed model shape begins.
3. From the main menu bar, select **View->Parallel** to turn off perspective.
4. In the context bar, click  to pause the animation after a full cycle has been completed.
5. In the context bar, click  and then select a node on the foam packaging near one of the corners that impacts the floor. When you restart the animation the camera will move with the selected node. If you zoom in on the node, it will remain in view throughout the animation.

    **Note:**

   To reset the camera to follow the global coordinate system, click  in the context bar.

While you view the deformation history of the drop test, take note of when the foam is in contact with the floor. You should observe that the initial impact occurs over the first 4 ms of the analysis. A second impact occurs from about 8 ms to 15 ms. The deformed state of the foam and board at approximately 4 ms after impact is shown in *Figure 1*.

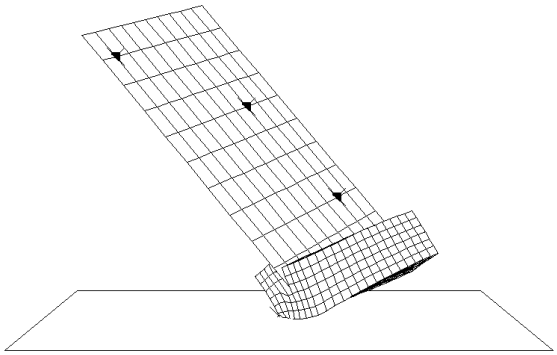


Figure 1: Deformed mesh at 4 ms.

Plotting model energy histories

Plot graphs of various energy variables versus time. Energy output can help you evaluate whether an Abaqus/Explicit simulation is predicting an appropriate response.

To plot energy histories:

1. In the Results Tree, click mouse button 3 on **History Output** for the output database named `Circuit.odb`. From the menu that appears, select **Filter**.
2. In the filter field, enter `*ALL*` to restrict the history output to just the energy output variables.
3. Select the ALLAE output variable, and save the data as `Artificial Energy`.
4. Select the ALLIE output variable, and save the data as `Internal Energy`.
5. Select the ALLKE output variable, and save the data as `Kinetic Energy`.
6. Select the ALLPD output variable, and save the data as `Plastic Dissipation`.
7. Select the ALLSE output variable, and save the data as `Strain Energy`.
8. In the Results Tree, expand the **XYData** container.
9. Select all five curves. Click mouse button 3, and select **Plot** from the menu that appears to view the *X*–*Y* plot.

   Next, you will customize the appearance of the plot; begin by changing the line styles of the curves.
10. Open the **Curve Options** dialog box.
11. In this dialog box, apply different line styles and thicknesses to each of the curves displayed in the viewport.

    Next, reposition the legend so that it appears inside the plot.
12. Double-click the legend to open the **Chart Legend Options** dialog box.
13. In this dialog box, switch to the **Area** tabbed page, and toggle on **Inset**.
14. In the viewport, drag the legend over the plot.

    Now change the format of the *X*-axis labels.
15. In the viewport, double-click the *X*-axis to access the **X Axis** options in the **Axis Options** dialog box.
16. In this dialog box, switch to the **Axes** tabbed page, and select the **Engineering** label format for the *X*-axis.

The energy histories appear as shown in *Figure 2*.

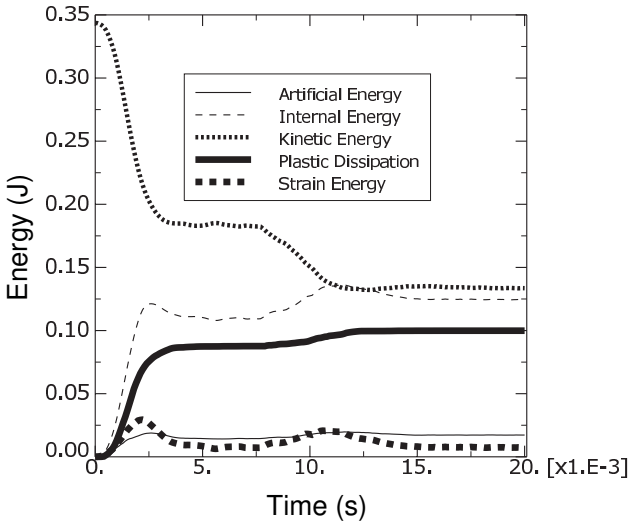


Figure 2: Energy results versus time.

First, consider the kinetic energy history. At the beginning of the simulation the components are in free fall, and the kinetic energy is large. The initial impact deforms the foam packaging, thus reducing the kinetic energy. The components then bounce and rotate about the impacted corner until the opposite side of the foam packaging impacts the floor at about 8 ms, further reducing the kinetic energy.

The deformation of the foam packaging during impact causes a transfer of kinetic energy to internal energy in the foam packaging and the circuit board. From *Figure 2* we can see that the internal energy increases as the kinetic energy decreases. In fact, the internal energy is composed of elastic energy and plastically dissipated energy, both of which are also plotted in *Figure 2*. Elastic energy rises to a peak and then falls as the elastic deformation recovers, but the plastically dissipated energy continues to rise as the foam is deformed permanently.

Another important energy output variable is the artificial energy, which is a substantial fraction (approximately 15%) of the internal energy in this analysis. By now you should know that the quality of the solution would improve if the artificial energy could be decreased to a smaller fraction of the total internal energy.

What causes high artificial strain energy in this problem?

Contact at a single node—such as the corner impact in this example—can cause hourglassing, especially in a coarse mesh. Two common strategies for reducing the artificial strain energy are to refine the mesh or to round the impacting corner. For the current exercise, however, we shall continue with the original mesh, realizing that improving the mesh would lead to an improved solution.

Evaluating acceleration histories at the chips

The next result we wish to examine is the acceleration of the chips attached to the circuit board. Excessive accelerations during impact may damage the chips. Therefore, in order to assess the desirability of the foam packaging, we need to plot the acceleration histories of the three chips. Since we expect the accelerations to be greatest in the 3-direction, we will plot the variable A3.

To plot acceleration histories:

1. In the Results Tree, filter the **History Output** container according to `*A3*`; select the acceleration `A3` of the nodes in sets `TopChip`, `MidChip`, and `BotChip`; and plot the three *X*–*Y* data objects.

The *X*–*Y* plot appears in the viewport. As before, customize the plot appearance to obtain a plot similar to *Figure 3*.

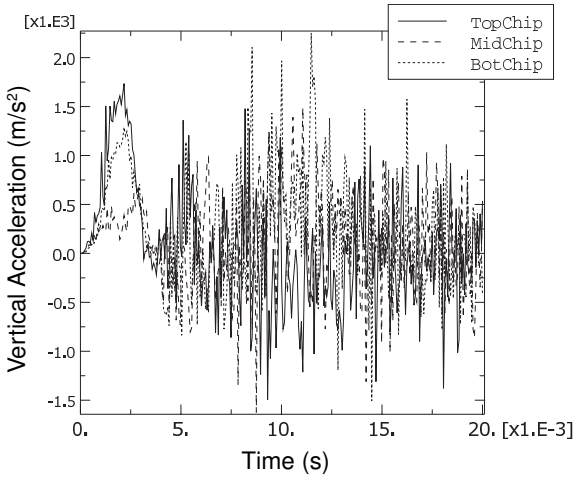


Figure 3: Acceleration of the three chips in the *Z*-direction.

Next, we will evaluate the plausibility of the acceleration data recorded at the bottom chip. To do this, we will integrate the acceleration data to calculate the chip velocity and displacement and compare the results to the velocity and displacement data recorded directly by Abaqus/Explicit.

To integrate the bottom chip acceleration history:

1. In the Results Tree, filter the **History Output** container according to `*BOTCHIP*`, select the acceleration `A3` of the node in set `BotChip`, and save the data as `A3`.
2. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
3. In the **Operate on XY Data** dialog box, integrate acceleration `A3` to calculate velocity and subtract the initial velocity magnitude of 4.43 m/s. The expression at the top of the dialog box should appear as:

   ```
   integrate ( "A3" ) - 4.43
   ```

4. Click **Plot Expression** to plot the calculated velocity curve.
5. In the Results Tree, click mouse button 3 on the velocity `V3` history output for the node in set `BotChip`; and select **Add to Plot** from the menu that appears.

   The *X*–*Y* plot appears in the viewport. As before, customize the plot appearance to obtain a plot similar to *Figure 4*. The velocity curve you produced by integrating the acceleration data may be different from the one pictured here. The reason for this will be discussed later.

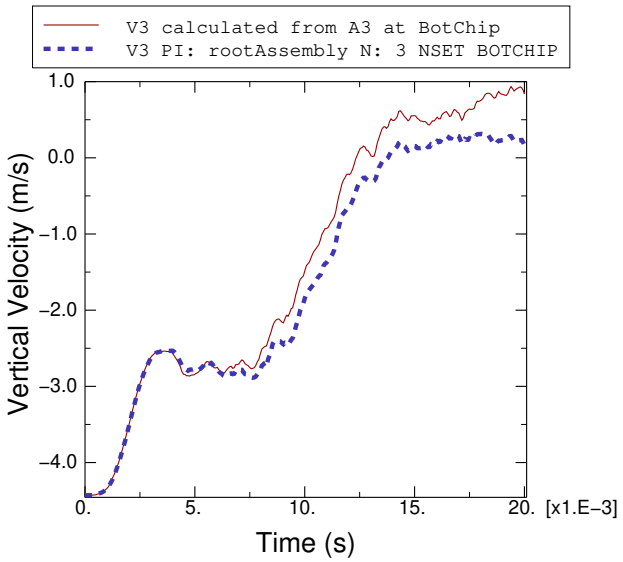


Figure 4: Velocity of the bottom chip in the *Z*-direction.

6. In the **Operate on XY Data** dialog box, integrate acceleration `A3` a second time to calculate chip displacement. The expression at the top of the dialog box should appear as:

   ```
   integrate ( integrate ( "A3" ) - 4.43 )
   ```

7. Click **Plot Expression** to plot the calculated displacement curve.

   Notice that the *Y*-value type is length. In order to plot the calculated displacement with the same *Y*-axis as the displacement output recorded during the analysis, we must save the *X*–*Y* data and change the *Y*-value type to displacement.

8. Click **Save As** to save the calculated displacement curve as `U3-from-A3`.

9. In the **XYData** container of the Results Tree, click mouse button 3 on `U3-from-A3`; and select **Edit** from the menu that appears.

10. In the **Edit XY Data** dialog box, choose **Displacement** as the *Y*-value type.

11. In the Results Tree, double-click `U3-from-A3` to recreate the calculated displacement plot with the displacement *Y*-value type.

12. In the Results Tree, click mouse button 3 on the displacement `U3` history output for the node in set `BotChip`; and select **Add to Plot** from the menu that appears.

    The *X*–*Y* plot appears in the viewport. As before, customize the plot appearance to obtain a plot similar to *Figure 5*. Again, the curve you produced by integrating the acceleration data may be different from the one pictured here. The reason for this will be discussed later.

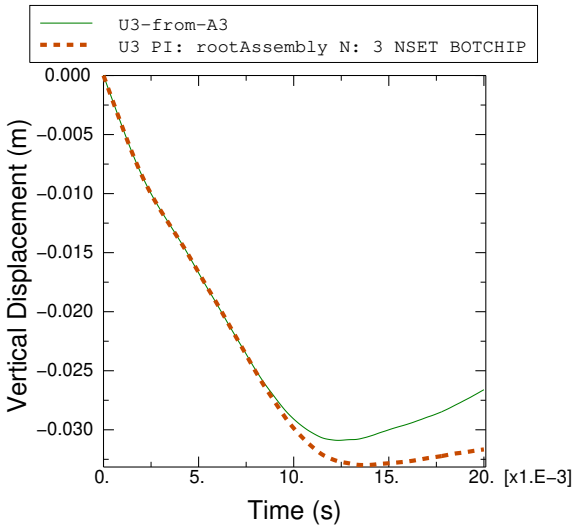


Figure 5: Displacement of the bottom chip in the *Z*-direction.

Why are the velocity and displacement curves calculated by integrating the acceleration data different from the velocity and displacement recorded during the analysis?

In this example the acceleration data has been corrupted by a phenomenon called aliasing. Aliasing is a form of data corruption that occurs when a signal (such as the results of an Abaqus analysis) is sampled at a series of discrete points in time, but not enough data points are saved in order to correctly describe the signal. The aliasing phenomenon can be addressed using digital signal processing (DSP) methods, a fundamental principle of which is the Nyquist Sampling Theorem (also known as the Shannon Sampling Theorem). The Sampling Theorem requires that a signal be sampled at a rate that is greater than twice the signal's highest frequency. Therefore, the maximum frequency content that can be described by a given sampling rate is half that rate (the Nyquist frequency). Sampling (storing) a signal with large-amplitude oscillations at frequencies greater than the Nyquist frequency of the sample rate may produce significantly distorted results due to aliasing. In this example the chip acceleration was sampled every 0.07 ms, which is a sampling rate of 14.3 kHz (the sample rate is the inverse of the sample size). The recorded data was aliased because the chip acceleration response has frequency content above 7.2 kHz (half the sample rate).

Aliasing of a sine wave

To better understand how aliasing distorts data, consider a 1 kHz sine wave sampled using various sampling rates, as shown in *Figure 6*.

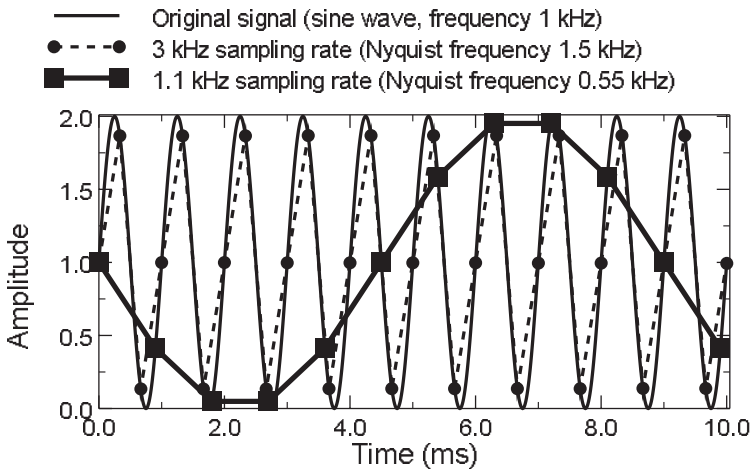


Figure 6: 1 kHz sine wave sampled at 1.1 kHz and 3 kHz.

According to the Sampling Theorem, this signal must be sampled at a rate greater than 2 kHz to avoid alias distortions. We will evaluate what happens when the sample rate is greater than or less than this value.

Consider the data recorded with a sample rate of 1.1 kHz; this rate is less than the required 2 kHz rate. The resulting curve exhibits alias distortions because it is an extremely misleading representation of the original 1 kHz sine wave.

Now consider the data recorded with a sample rate of 3 kHz; this rate is greater than the required 2 kHz rate. The frequency content of the original signal is captured without aliasing. However, this sample rate is not high enough to guarantee that the peak values of the sampled signal are captured very accurately. To guarantee 95% accuracy of the recorded local peak values, the sampling rate must exceed the signal frequency by a factor of ten or more.

Avoiding aliasing

In the previous two examples of aliasing (the aliased chip acceleration and the aliased sine wave), it would not have been obvious from the aliased data alone that aliasing had occurred. In addition, there is no way to uniquely reconstruct the original signal from the aliased data alone. Therefore, care should be taken to avoid aliasing your analysis results, particularly in situations when aliasing is most likely to occur.

Susceptibility to aliasing depends on a number of factors, including output rate, output variable, and model characteristics. Recall that signals with large-amplitude oscillations at frequencies greater than half the sampling rate (the Nyquist frequency) may be significantly distorted due to aliasing. The two output variables that are most likely to have large-amplitude high-frequency content are accelerations and reaction forces. Therefore, these variables are the most susceptible to aliasing. Displacements, on the other hand, are lower in frequency content by nature, so they are much less susceptible to aliasing. Other result variables, such as stress and strain, fall somewhere in between these two extremes. Any model characteristic that reduces the high-frequency response of the solution will decrease the analysis's susceptibility to aliasing. For example, an elastically dominated impact problem would be even more susceptible to aliasing than this circuit board drop test which includes energy absorbing packaging.

The safest way to ensure that aliasing is not a problem in your results is to request output at every increment. When you do this, the output rate is determined by the stable time increment, which is based on the highest possible frequency response of the model. However, requesting output at every increment is often not practical because it would result in very large output files. In addition, output at every increment is usually much more data than you need; there is no need to capture high-frequency solution noise when what you are really interested in is the lower-frequency structural response. An alternative method for avoiding aliasing is to request output at

a lower rate and use the Abaqus/Explicit real-time filtering capabilities to remove high-frequency content from the result before writing data to the output database file. This technique uses less disk space than requesting output every increment; however, it is up to you to ensure that your output rate and filter choices are appropriate (to avoid aliasing or other distortions related to digital signal processing).

Abaqus/Explicit offers filtering capabilities for both field and history data. Filtering of history data only is discussed here.

Rerunning the analysis with output filtering

In this section you will add real-time filters to the history output requests for the circuit board drop test analysis. While Abaqus/Explicit does allow you to create user-defined output filters (Butterworth, Chebyshev Type I, and Chebyshev Type II) based on criteria that you specify, in this example we will use the built-in antialiasing filter. The built-in antialiasing filter is designed to give you the best un-aliased representation of the results recorded at the output rate you specify on the output request. To do this, Abaqus/Explicit internally applies a low-pass, second-order, Butterworth filter with a cutoff frequency set to one-sixth of the sampling rate. For more information, see "Overview of filtering Abaqus history output" in the Dassault Systèmes Knowledge Base at *https://www.3ds.com/support/knowledge-base*. For more information on defining your own real-time filters, see *Filtering Output and Operating on Output in Abaqus/Explicit*.

Modifying the history output requests

When Abaqus writes nodal history output to the output database, it gives each data object a name that indicates the recorded output variable, the filter used (if any), the part instance name, the node number, and the node set. For this exercise you will be creating multiple output requests for the node in set `BotChip` that differ only by the output sample rate, which is not a component of the history output name. To easily distinguish between the similar output requests, create two new sets for the bottom chip reference node. Name one of the new sets `BotChip-all` and the other `BotChip-largeInc`.

Next, copy the history output request for the bottom chip three times. Edit the first copy to activate the **Antialiasing** filter; for this request continue to record data every 7×10^{-5} s using the set `BotChip`. Edit the second copy to record the data at every time increment; apply this output request to the set `BotChip-all`. Edit the third copy to record the data at every 7×10^{-4} s; apply this output request to the set `BotChip-largeInc` and activate the **Antialiasing** filter. When you are finished, there will be four history output requests for the bottom chip (the original one and the three added here).

Edit the output request for the strains in the set `BotBoard` in order to activate the **Antialiasing** filter.

Although we will not be discussing the results here, you may wish to add the **Antialiasing** filter to the history output request for the displacement, velocity, and acceleration of the node sets `MidChip` and `TopChip`.

Save your model database, and submit the job for analysis.

Evaluating the filtered acceleration of the bottom chip

When the analysis completes, test the plausibility of the acceleration history output for the bottom chip recorded every 0.07 ms using the built-in, antialiasing filter. Do this by saving and then integrating the filtered acceleration data (`A3_ANTIALIASING` for set `BotChip`) and comparing the results to recorded velocity and displacement data, just as you did earlier for the unfiltered version of these results. This time you should find that the velocity and displacement curves calculated by integrating the filtered acceleration are very similar to the velocity and displacement values written to the output database during the analysis. You may also have noticed that the velocity and displacement results are the same regardless of whether or not the built-in antialiasing filter is used. This is because the highest frequency content of the nodal velocity and displacement curves is much less than half the sampling rate. Consequently, no aliasing occurred when the data was recorded without filtering, and when the built-in antialiasing filter was applied it had no effect because there was no high frequency response to remove.

Next, compare the acceleration `A3` history output recorded every increment with the two acceleration `A3` history curves recorded every 0.07 ms. Plot the data recorded at every increment first so that it does not obscure the other results.

To plot the acceleration histories

1. In the Results Tree, filter the **History Output** container according to `*A3*BOTCHIP*` and double-click the acceleration `A3` history output for the node set `BotChip-all`.
2. Select the two acceleration `A3` history output objects for the node set `BotChip` (one filtered with the built-in antialiasing filter and the other with no filtering) using **[Ctrl] + Click**; click mouse button 3 and select **Add to Plot** from the menu that appears.

   The *X*–*Y* plot appears in the viewport. Zoom in to view only the first third of the results and customize the plot appearance to obtain a plot similar to *Figure 1*.

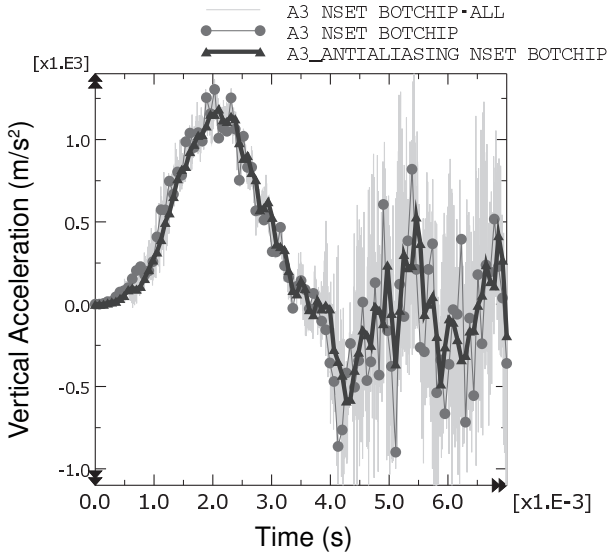


Figure 1: Comparison of acceleration output with and without filtering.

First consider the acceleration history recorded every increment. This curve contains a lot of data, including high-frequency solution noise which becomes so large in magnitude that it obscures the structurally-significant lower-frequency components of the acceleration. When output is requested every increment, the output time increment is the same as the stable time increment, which (in order to ensure stability) is based on a conservative estimate of the highest possible frequency response of the model. Frequencies of structural significance are typically two to four orders of magnitude less than the highest frequency of the model. In this example the stable time increment ranges between 8.4×10^{-4} ms to 8.8×10^{-4} ms (see the status file, `Circuit.sta`), which corresponds to a sample rate of about 1 MHz; this sample rate has been rounded down for this discussion, even though it means that the value is not conservative. Recalling the Sampling Theorem, the highest frequency that can be described by a given sample rate is half that rate; therefore, the highest frequency of this model is about 500 kHz and typical structural frequencies could be as high as 2–3 kHz (more than 2 orders of magnitude less than the highest model frequency). While the output recorded every increment contains a lot of undesirable solution noise in the 3 to 500 kHz range, it is guaranteed to be good (not aliased) data, which can be filtered later with a postprocessing operation if necessary.

Next consider the data recorded every 0.07 ms without any filtering. Recall that this is the curve we know to be corrupted by aliasing. The curve jumps from point to point by directly including whatever the raw acceleration value happens to be after each 0.07 ms interval. The variable nature of the high-frequency noise makes this aliased result very sensitive to otherwise imperceptible variations in the solution (due to differences between computer platforms, for example), hence the results you recorded every 0.07 increments may be significantly different from those shown in *Figure 1*. Similarly, the velocity and displacement curves we produced by integrating

the aliased acceleration (*Figure 4* and *Figure 5*) data are extremely sensitive to small differences in the solution noise.

When the built-in antialiasing filter is applied to the output requested every 0.07 ms, frequency content that is too high to be captured by the 14.3 kHz sample rate is filtered out before the result is written to the output database. To do this, Abaqus internally defines a low-pass, second-order, Butterworth filter. Low-pass filters attenuate the frequency content of a signal that is above a specified cutoff frequency. An ideal low-pass filter would completely eliminate all frequencies above the cutoff frequency while having no effect on the frequency content below the cutoff frequency. In reality there is a transition band of frequencies surrounding the cutoff frequency that are partially attenuated. To compensate for this, the built-in antialiasing filter has a cutoff frequency that is one-sixth of the sample rate, a value lower than the Nyquist frequency of one-half the sample rate. In most cases (including this example), this cutoff frequency is adequate to ensure that all frequency content above the Nyquist frequency has been removed before the data are written to the output database.

Abaqus/Explicit does not check to ensure that the specified output time interval provides an appropriate cutoff frequency for the internal antialiasing filter; for example, Abaqus does not check that only the noise of the signal is eliminated. When the acceleration data are recorded every 0.07 ms, the internal antialiasing filter is applied with a cutoff frequency of 2.4 kHz. This cutoff frequency is nearly the same value we previously determined to be the maximum physically meaningful frequency for the model (more than two orders of magnitude less than the maximum frequency the stable time increment can capture). The 0.07 ms output interval was intentionally chosen for this example to avoid filtering frequency content that could be physically meaningful. Next, we will study the results when the antialiasing filter is applied with a sample interval that is too large.

To plot the filtered acceleration histories

1. In the Results Tree, double-click the acceleration `A3` history output for the node set `BotChip-all`.
2. Select the two filtered acceleration `A3_ANTIALIASING` history output objects for the bottom chip; click mouse button 3 and select **Add to Plot** from the menu that appears.

   The *X*–*Y* plot appears in the viewport. Zoom out and customize the plot appearance to obtain a plot similar to *Figure 2*.

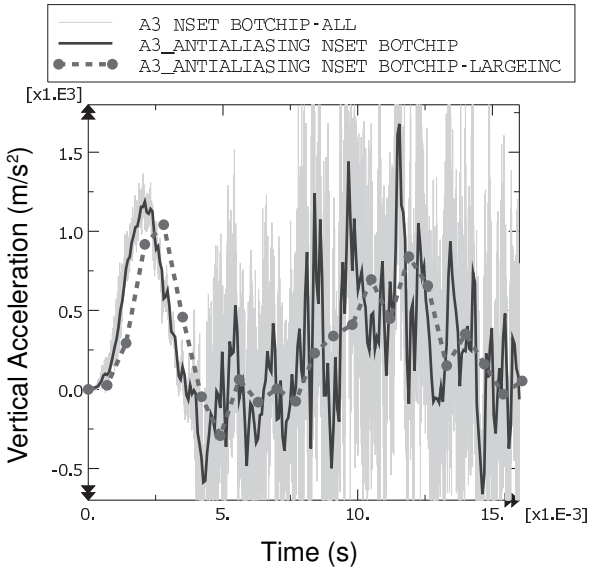


Figure 2: Filtered acceleration with different output sampling rates.

Figure 2 clearly illustrates some of the problems that can arise when the built-in antialiasing filter is used with too large an output time increment. First, notice that many of the oscillations in the acceleration output are filtered

out when the acceleration is recorded with large time increments. In this dynamic impact problem it is likely that a significant portion of the removed frequency content is physically meaningful. Previously, we estimated that the frequency of the structural response may be as large as 2–3 kHz; however, when the sample interval is 0.7 ms, filtering is performed with a low cutoff frequency of 0.24 kHz (sample interval of 0.7 ms corresponds to a sample frequency of 1.43 kHz, one-sixth of which is the 0.24 kHz cutoff frequency). Even though the results recorded every 0.7 ms may not capture all physically meaningful frequency content, it does capture the low-frequency content of the acceleration data without distortions due to aliasing. Keep in mind that filtering decreases the peak value estimations, which is desirable if only solution noise is filtered, but can be misleading when physically meaningful solution variations have been removed.

Another issue to note is that there is a time delay in the acceleration results recorded every 0.7 ms. This time delay (or phase shift) affects all real-time filters. The filter must have some input in order to produce output; consequently the filtered result will include some time delay. While some time delay is introduced for all real-time filtering, the time delay becomes more pronounced as the filter cutoff frequency decreases; the filter must have input over a longer span of time in order to remove lower frequency content. Increasing the filter order (an option if you have created a user-defined filter, rather than using the second-order built-in antialiasing filter) also results in an increase in the output time delay. For more information, see *Filtering Output and Operating on Output in Abaqus/Explicit*.

Use the real-time filtering functionality with caution. In this example we would not have been able to identify the problems with the heavily filtered data if we did not have appropriate data for comparison. In general it is best to use a minimal amount of filtering in Abaqus/Explicit, so that the output database contains a rich, un-aliased, representation for the solution recorded at a reasonable number of time points (rather than at every increment). If additional filtering is necessary, it can be done as a postprocessing operation in Abaqus/CAE.

Filtering acceleration history in Abaqus/CAE

In this section we will use the Visualization module in Abaqus/CAE to filter the acceleration history data written to the output database. Filtering as a postprocessing operation in the Visualization module has several advantages over the real-time filtering available in Abaqus/Explicit. In the Visualization module you can quickly filter *X–Y* data and plot the results. You can easily compare the filtered results to the unfiltered results to verify that the filter produced the desired effect. Using this technique you can quickly iterate to find appropriate filter parameters. In addition, the Visualization module filters do not suffer from the time delay that is unavoidable when filtering is applied during the analysis. Keep in mind, however, that postprocessing filters cannot compensate for poor analysis history output; if the data has been aliased or if physically meaningful frequencies have been removed, no postprocessing operation can recover the lost content.

To demonstrate the differences between filtering in the Visualization module and filtering in Abaqus/Explicit, we will filter the acceleration of the bottom chip in the Visualization module and compare the results to the filtered data Abaqus/Explicit wrote to the output database.

To filter acceleration history:

1. In the Results Tree, select the acceleration `A3` history output for the node set `BotChip-all`, and save the data as `A3-all`.
2. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
3. In the **Operate on XY Data** dialog box, filter `A3-all` with filter options that are equivalent to those applied by the Abaqus/Explicit built-in antialiasing filter when the output increment is 0.7 ms. Recall that the built-in antialiasing filter is a second-order Butterworth filter with a cutoff frequency that is one-sixth of the output sample rate; hence, the expression at the top of the dialog box should appear as

```
butterworthFilter ( xyData="A3-all",
 cutoffFrequency=1/(6*0.0007) )
```

4. Click **Plot Expression** to plot the filtered acceleration curve.
5. In the Results Tree, click mouse button 3 on the filtered acceleration `A3_ANTIALIASING` history output for node set `BotChip-largeInc`; and select **Add to Plot** from the menu that appears. If you wish, also add the filtered acceleration history for the node set `BotChip`.

   The *X–Y* plot appears in the viewport. As before, customize the plot appearance to obtain a plot similar to *Figure 3*.

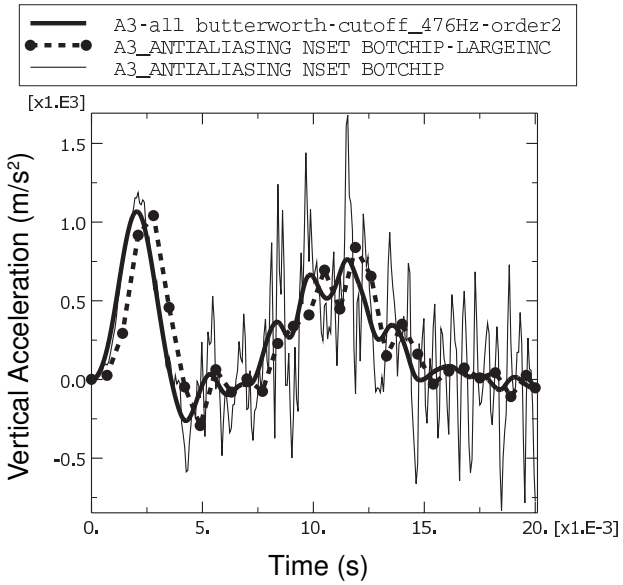


Figure 3: Comparison of acceleration filtered in Abaqus/Explicit and the Visualization module.

In *Figure 3* it is clear that the postprocessing filter in the Visualization module of Abaqus/CAE does not suffer from the time delay that occurs when filtering is performed while the analysis is running. This is because the Visualization module filters are bidirectional, which means that the filtering is applied first in a forward pass (which introduces some time delay) and then in a backward pass (which removes the time delay). As a consequence of the bidirectional filtering in the Visualization module, the filtering is essentially applied twice, which results in additional attenuation of the filtered signal compared to the attenuation achieved with a single-pass filter. This is why the local peaks in the acceleration curve filtered in the Visualization module are a bit lower than those in the curve filtered by Abaqus/Explicit.

To develop a better understanding of the Visualization module filtering capabilities, return to the **Operate on XY Data** dialog box and filter the acceleration data with other filter options. For example, try different cutoff frequencies.

Can you confirm that the cutoff frequency of 2.4 kHz associated with the built-in antialiasing filter with a time increment size of 0.07 was appropriate? Does increasing the cutoff frequency to 6 kHz, 7 kHz, or even 10 kHz produce significantly different results?

You should find that a moderate increase in the cutoff frequency does not have a significant effect on the results, implying that we probably have not missed physically meaningful frequency content when we filtered with a cutoff frequency of 2.4 kHz.

Compare the results of filtering the acceleration data with Butterworth and Chebyshev Type I filters. The Chebyshev filter requires a ripple factor parameter (*rippleFactor*), which indicates how much oscillation you will allow in exchange for an improved filter response; see *Filtering Output and Operating on Output in Abaqus/Explicit* for more information. For the Chebyshev Type I filter a ripple factor of 0.071 will result in a very flat pass band with a ripple that is only 0.5%.

You may not notice much difference between the filters when the cutoff frequency is 5 kHz, but what about when the cutoff frequency is 2 kHz? What happens when you increase the order of the Chebyshev Type I *filter?*

Compare your results to those shown in *Figure 4*.

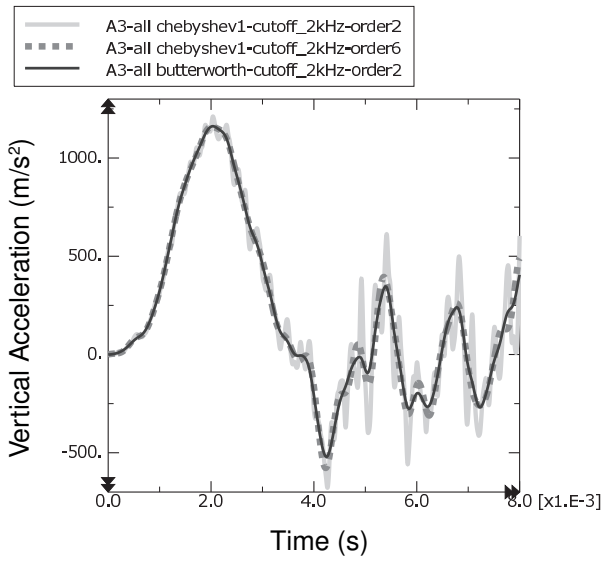


Figure 4: Comparison of acceleration filtered with Butterworth and Chebyshev Type I filters.

Note:

The Abaqus/CAE postprocessing filters are second-order by default. To define a higher order filter you can use the *filterOrder* parameter with the **butterworthFilter** and the **chebyshev1Filter** operators. For example, use the following expression in the **Operate on XY Data** dialog box to filter `A3-all` with a sixth-order Chebyshev Type I filter using a cutoff frequency of 2 kHz and a ripple factor of 0.017.

```
chebyshev1Filter ( xyData="A3-all" , cutoffFrequency=2000,
 rippleFactor= 0.017, filterOrder=6)
```

The second-order Chebyshev Type I filter with a ripple factor of 0.071 is a relatively weak filter, so some of the frequency content above the 2 kHz cutoff frequency is not filtered out. When the filter order is increased, the filter response is improved so that the results are more like the equivalent Butterworth filter. For more information on the *X–Y* data filters available in Abaqus/CAE see *Operating on saved X–Y data objects*.

Filtering strain history in Abaqus/CAE

Strain in the circuit board near the location of the chips is another result that may assist us in determining the effectiveness of the foam packaging. If the strain under the chips exceeds a limiting value, the solder securing the chips to the board will fail. We wish to identify the peak strain in any direction. Therefore, the maximum and minimum principal logarithmic strains are of interest. Principal strains are one of a number of Abaqus results that are derived from nonlinear operators; in this case a nonlinear function is used to calculate principal strains from the individual strain components. Some other common results that are derived from nonlinear operators are principal stresses, Mises stress, and equivalent plastic strains. Care must be taken when filtering results that are derived from nonlinear operators, because nonlinear operators (unlike linear ones) can modify the frequency of the original result. Filtering such a result may have undesirable consequences; for example, if you remove a portion of the frequency content that was introduced by the application of the nonlinear operator, the filtered result will be a distorted representation of the derived quantity. In general, you should either avoid filtering quantities derived from nonlinear operators or filter the underlying quantities before calculating the derived quantity using the nonlinear operator.

The strain history output for this analysis was recorded every 0.07 ms using the built-in antialiasing filter. To verify that the antialiasing filter did not distort the principal strain results, we will calculate the principal logarithmic strains using the filtered strain components and compare the result to the filtered principal logarithmic strains.

To calculate the principal logarithmic strains:

1. To identify the elements in set `BotBoard` that are closest to the bottom chip (use the ODB display options to display the mass elements), plot the undeformed circuit board with element numbers visible.
2. In the Results Tree, filter the **History Output** according to `*LE*Element #*`, where # is the number of one of the elements in set `BotBoard` that is close to the bottom chip. Select the logarithmic strain component `LE11` on the `SPOS` surface of the element, and save the data as `LE11`.
3. Similarly, save the `LE12` and `LE22` strain components for the same element as `LE12` and `LE22`, respectively.
4. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
5. In the **Operate on XY Data** dialog box, use the saved logarithmic strain components to calculate the maximum principal logarithmic strain. The expression at the top of the dialog box should appear as:

   ```
   (("LE11"+"LE22")/2) + sqrt( power(("LE11"-"LE22")/2,2)
    + power("LE12"/2,2) )
   ```

6. Click **Save As** to save calculated maximum principal logarithmic strain as `LEP-Max`.
7. Edit the expression in the **Operate on XY Data** dialog box to calculate the minimum principal logarithmic strain. The modified expression should appear as:

   ```
   (("LE11"+"LE22")/2) - sqrt( power(("LE11"-"LE22")/2,2)
    + power("LE12"/2,2) )
   ```

8. Click **Save As** to save calculated minimum principal logarithmic strain as `LEP-Min`.

   In order to plot the calculated principal logarithmic strains with the same *Y*-axis as the strains recorded during the analysis, change the *Y*-value type to strain.
9. In the **XYData** container of the Results Tree, click mouse button 3 on `LEP-Max`; and select **Edit** from the menu that appears.
10. In the **Edit XY Data** dialog box, choose **Strain** as the *Y*-value type.
11. Similarly, edit `LEP-Min` and select **Strain** as the *Y*-value type.
12. Using the Results Tree, plot `LEP-Man` and `LEP-Min` along with the principal strains recorded during the analysis (`LEP1` and `LEP2`) for the same element in set `BotBoard`.
13. As before, customize the plot appearance to obtain a plot similar to *Figure 5*. The actual plot will depend on which element you selected.

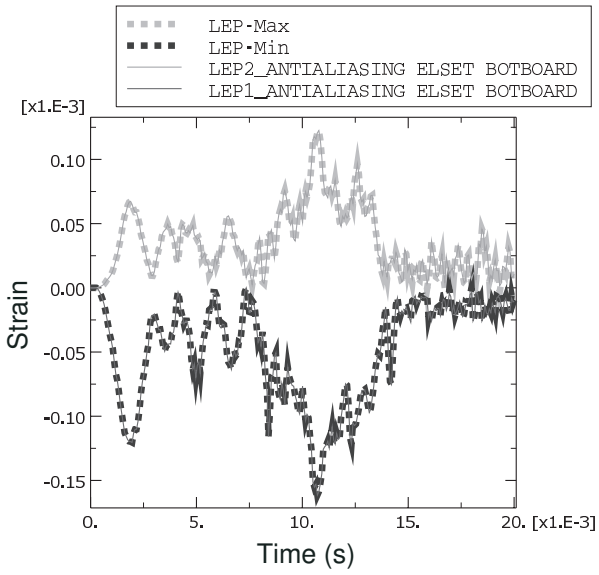


Figure 5: Principal logarithmic strain values versus time.

In *Figure 5* we see that the filtered principal logarithmic strain curves recorded during the analysis are indistinguishable from the principal logarithmic strain curves calculated from the filtered strain components. Therefore the antialiasing filter (cutoff frequency 2.4 kHz) did not remove any of the frequency content introduced by the nonlinear operation to calculate principal strains form the original strain data. Next, filter the strain data with a lower cutoff frequency of 500 Hz.

To filter principal logarithmic strains with a cutoff frequency of 500 Hz:

1. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
2. In the **Operate on XY Data** dialog box, filter the maximum principal logarithmic strain `LEP-Max` using a second-order Butterworth filter with a cutoff frequency of 500 Hz. The expression at the top of the dialog box should appear as:

   ```
   butterworthFilter(xyData="LEP-Max", cutoffFrequency=500)
   ```

3. Click **Save As** to save the calculated maximum principal logarithmic strain as `LEP-Max-FilterAfterCalc-bw500`.
4. Similarly, filter the logarithmic strain components `LE11`, `LE12`, and `LE22` using the same second-order Butterworth filter with a cutoff frequency of 500 Hz. Save the resulting curves as `LE11-bw500`, `LE12-bw500`, and `LE22-bw500`, respectively.
5. Now calculate the maximum principal logarithmic strain using the filtered logarithmic strain components. The expression at the top of the **Operate on XY Data** dialog box should appear as:

   ```
   (("LE11-bw500"+"LE22-bw500")/2) + sqrt(
    power(("LE11-bw500"-"LE22-bw500")/2,2) +
    power("LE12-bw500"/2,2) )
   ```

6. Click **Save As** to save the calculated maximum principal logarithmic strain as `LEP-Max-CalcAfterFilter-bw500`.
7. In the **XYData** container of the Results Tree, click mouse button 3 on `LEP-Max-CalcAfterFilter-bw500`; and select **Edit** from the menu that appears.
8. In the **Edit XY Data** dialog box, choose **Strain** as the *Y*-value type.

9. Plot `LEP-Max-CalcAfterFilter-bw500` and `LEP-Max-FilterAfterCalc-bw500` as shown in *Figure 6*. As before, the actual plot will depend on which element you selected.

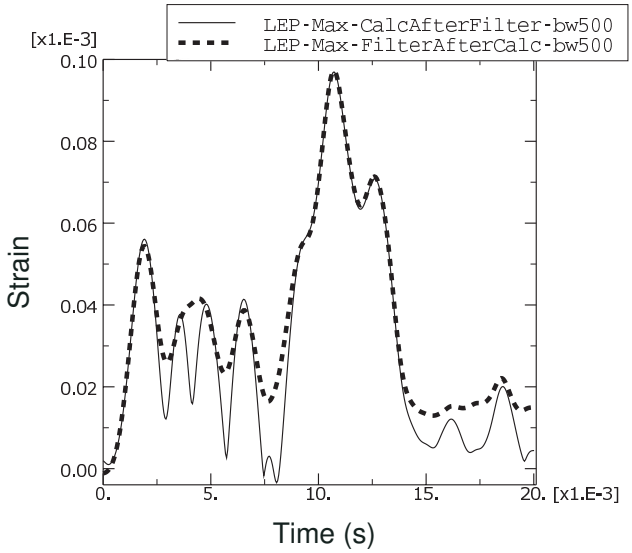


Figure 6: Principal logarithmic strain calculated before and after filtering (cutoff frequency 500 Hz).

In *Figure 6* you can see that there is a significant difference between filtering the strain data before and after the principal strain calculation. The curve that was filtered after the principal strain calculation is distorted because some of the frequency content introduced by applying the nonlinear principal-stress operator is higher than the 500 Hz filter cutoff frequency. In general, you should avoid directly filtering quantities that have been derived from nonlinear operators; whenever possible filter the underlying components and then apply the nonlinear operator to the filtered components to calculate the desired derived quantity.

Strategy for recording and filtering Abaqus/Explicit history output

Recording output for every increment in Abaqus/Explicit generally produces much more data than you need. The real-time filtering capability allows you to request history output less frequently without distorting the results due to aliasing. However, you should ensure that your output rate and filtering choices have not removed physically meaningful frequency content nor distorted the results (for example, by introducing a large time delay or by removing frequency content introduced by nonlinear operators). Keep in mind that no amount of postprocessing filtering can recover frequency content filtered out during the analysis, nor can postprocessing filtering recover an original signal from aliased data. In addition, it may not be obvious when results have been over-filtered or aliased if additional data are not available for comparison. A good strategy is to choose a relatively high output rate and use the Abaqus/Explicit filters to prevent aliasing of the history output, so that valid and rich results are written to the output database. You may even wish to request output at every increment for a couple of critical locations. After the analysis completes, use the postprocessing tools in Abaqus/CAE to quickly and iteratively apply additional filtering as desired.

Compatibility between Abaqus/Standard and Abaqus/Explicit

There are fundamental differences in the mechanical contact algorithms in Abaqus/Standard and Abaqus/Explicit. These differences are reflected in how contact conditions are defined.
The main differences are the following:

- For contact pairs Abaqus/Standard typically uses a pure main-secondary relationship for the contact constraints by default (see *About Contact Pairs in Abaqus/Standard*); the nodes of the secondary surface are constrained not to penetrate into the main surface. The nodes of the main surface can, in principle, penetrate into the secondary surface. Abaqus/Explicit includes this formulation but typically uses a balanced main-secondary weighting by default (see *Contact Formulations for Contact Pairs in Abaqus/Explicit*).
- The contact formulations in Abaqus/Standard and Abaqus/Explicit differ in many respects due to different convergence, performance, and numerical requirements.
- The constraint enforcement methods in Abaqus/Standard and Abaqus/Explicit differ in some respects. For example, both Abaqus/Standard and Abaqus/Explicit provide penalty constraint methods, but the default penalty stiffnesses differ.
- Abaqus/Standard and Abaqus/Explicit both provide a small-sliding contact formulation (see *Contact Formulations in Abaqus/Standard* and *Contact Formulations for Contact Pairs in Abaqus/Explicit*). However, the small-sliding contact formulation in Abaqus/Standard transfers the load to the main nodes according to the current position of the secondary node. Abaqus/Explicit always transfers the load through the anchor point.

As a result of these differences, contact definitions specified in an Abaqus/Standard analysis cannot be imported into an Abaqus/Explicit analysis and vice versa (see *Transferring Results between Abaqus/Explicit and Abaqus/Standard*).

Related Abaqus examples

Related Abaqus examples are available.

- *Indentation of a crushable foam plate*
- *Pressure penetration analysis of an air duct kiss seal*
- *Deep drawing of a cylindrical cup*

Suggested reading

The following references provide additional information on contact analysis with finite element methods. They allow the interested user to explore the topic in more depth.

General texts on contact analysis

- Belytschko, T., W. K. Liu, and B. Moran, *Nonlinear Finite Elements for Continua and Structures*, Wiley & Sons, 2000.
- Crisfield, M. A., *Non-linear Finite Element Analysis of Solids and Structures, Volume II: Advanced Topics*, Wiley & Sons, 1997.
- Johnson, K. L., *Contact Mechanics*, Cambridge, 1985.
- Oden, J. T., and G. F. Carey, *Finite Elements: Special Problems in Solid Mechanics,* Prentice-Hall, 1984.

General text on digital signal proccesing

- Stearns, S. D., and R. A. David, *Signal Processing Algorithms in MATLAB*, Prentice Hall P T R, 1996.

Summary

This section presents a summary of the concepts.

- Contact analyses require a careful, logical approach. Divide the analysis into several steps if necessary, and apply the loading slowly making sure that the contact conditions are well established.
- In general, it is best to use a separate step for each part of the analysis in Abaqus/Standard even if it is just to change boundary conditions to loads. You will almost certainly end up with more steps than anticipated, but the model should converge much more easily. Contact analyses are much more difficult to complete if you try to apply all the loads in one step.
- In Abaqus/Standard achieve stable contact conditions between all components before applying the working loads to the structure. If necessary, apply temporary boundary conditions, which may be removed at a later stage. The final results should be unaffected, provided that the constraints produce no permanent deformation.
- Do not apply boundary conditions to nodes on contact surfaces that constrain the node in the direction of contact in Abaqus/Standard. If there is friction, do not constrain these nodes in any degree of freedom: zero pivot messages may result.
- Always try to use first-order elements for contact simulations in Abaqus/Standard.
- Both Abaqus/Standard and Abaqus/Explicit provide two distinct algorithms for modeling contact: general contact and contact pairs.
- General contact interactions allow you to define contact between many or all regions of a model; contact pair interactions describe contact between two surfaces or between a single surface and itself.
- Surfaces used with the general contact algorithm can span multiple unattached bodies. More than two surface facets can share a common edge. In contrast, all surfaces used with the contact pair algorithm must be continuous and simply connected.
- In Abaqus/Explicit single-sided surfaces on shell, membrane, or rigid elements must be defined so that the normal directions do not “flip” as the surface is traversed.
- Abaqus/Explicit does not smooth rigid surfaces; they are faceted like the underlying elements. Coarse meshing of discrete rigid surfaces can produce noisy solutions with the contact pair algorithm. The general contact algorithm does include some numerical rounding of features.
- Tie constraints are a useful means of mesh refinement in Abaqus.
- Abaqus/Explicit adjusts the nodal coordinates without strain to remove any initial overclosures prior to the first step. If the adjustments are large with respect to the element dimensions, elements can become severely distorted.
- In subsequent steps any nodal adjustments to remove initial overclosures in Abaqus/Explicit induce strains that can potentially cause severe mesh distortions.
- When you are interested in results that are likely to contain high frequency oscillations, such as accelerations in an impact problem, request Abaqus/Explicit history output with a relatively high output rate and (if the output rate is less than every increment) apply an antialiasing filter; then, use a postprocessing filter if stronger filtering is desired.
- The *Abaqus Interactions Guide* contains more detailed discussions of contact modeling in Abaqus. *About Contact Interactions* is a good place to begin further reading on the subject.

Quasi-Static Analysis with Abaqus/Explicit

The explicit solution method is a true dynamic procedure originally developed to model high-speed impact events in which inertia plays a dominant role in the solution. Out-of-balance forces are propagated as stress waves between neighboring elements while solving for a state of dynamic equilibrium. Since the minimum stable time increment is usually quite small, most problems require a large number of increments.

The explicit solution method has proven valuable in solving quasi-static problems as well—Abaqus/Explicit solves certain types of static problems more readily than Abaqus/Standard does. One advantage of the explicit procedure over the implicit procedure is the greater ease with which it resolves complicated contact problems. In addition, as models become very large, the explicit procedure requires fewer system resources than the implicit procedure. Refer to *Comparison of implicit and explicit procedures* for a detailed comparison of the implicit and explicit procedures.

Applying the explicit dynamic procedure to quasi-static problems requires some special considerations. Since a static solution is, by definition, a long-time solution, it is often computationally impractical to simulate an event in its natural time scale, which would require an excessive number of small time increments. To obtain an economical solution, the event must be accelerated in some way. The problem is that as the event is accelerated, the state of static equilibrium evolves into a state of dynamic equilibrium in which inertial forces become more dominant. The goal is to model the process in the shortest time period in which inertial forces remain insignificant.

Quasi-static analyses can also be conducted in Abaqus/Standard. Quasi-static stress analysis in Abaqus/Standard is used to analyze linear or nonlinear problems with time-dependent material response (creep, swelling, viscoelasticity, and two-layer viscoplasticity) when inertia effects can be neglected. For more information on quasi-static analysis in Abaqus/Standard, see *Quasi-Static Analysis*.

In this section:

- *Analogy for explicit dynamics*
- *Loading rates*
- *Mass scaling*
- *Energy balance*
- *Example: forming a channel in Abaqus/Explicit*
- *Summary*

Analogy for explicit dynamics

To provide you with a more intuitive understanding of the differences between a slow, quasi-static loading case and a rapid loading case, we use the analogy illustrated in *Figure 1*.



Figure 1: Analogy for slow and fast loading cases.

The figure shows two cases of an elevator full of passengers. In the slow case the door opens and you walk in. To make room, the occupants adjacent to the door slowly push their neighbors, who push their neighbors, and so on. This disturbance passes through the elevator until the people next to the walls indicate that they cannot move. A series of waves pass through the elevator until everyone has reached a new equilibrium position. If you increase your speed slightly, you will shove your neighbors more forcefully than before, but in the end everyone will end up in the same position as in the slow case.

In the fast case the door opens and you run into the elevator at high speed, permitting the occupants no time to rearrange themselves to accommodate you. You will injure the two people directly in front of the door, while the other occupants will be unaffected.

The same thinking is true for quasi-static analyses. The speed of the analysis often can be increased substantially without severely degrading the quality of the quasi-static solution; the end result of the slow case and a somewhat accelerated case are nearly the same. However, if the analysis speed is increased to a point at which inertial effects dominate, the solution tends to localize, and the results are quite different from the quasi-static solution.

Loading rates

The actual time taken for a physical process is called its natural time.
Generally, it is safe to assume that performing an analysis in the natural time for a quasi-static process will produce accurate static results. After all, if the real-life event actually occurs in a natural time scale in which velocities are zero at the conclusion, a dynamic analysis should be able to capture the fact that the analysis has, in fact, achieved a steady state. You can increase the loading rate so that the same physical event occurs in less time as long as the solution remains nearly the same as the true static solution and dynamic effects remain insignificant.

In this section:

- *Smooth amplitude curves*
- *Structural problems*
- *Metal forming problems*

Smooth amplitude curves

For accuracy and efficiency quasi-static analyses require the application of loading that is as smooth as possible. Sudden, jerky movements cause stress waves, which can induce noisy or inaccurate solutions. Applying the load in the smoothest possible manner requires that the acceleration changes only a small amount from one increment to the next. If the acceleration is smooth, it follows that the changes in velocity and displacement are also smooth.

Abaqus has a simple, built-in smooth step amplitude curve that automatically creates a smooth loading amplitude. When you define a smooth step amplitude curve, Abaqus automatically connects each of your data pairs with curves whose first and second derivatives are smooth and whose values are zero at each of your data points. Since both of these derivatives are smooth, you can apply a displacement loading with a smooth step amplitude curve using only the initial and final data points, and the intervening motion will be smooth. Using this type of loading amplitude allows you to perform a quasi-static analysis without generating waves due to discontinuity in the rate of applied loading. An example of a smooth step amplitude curve is shown in *Figure 1*.

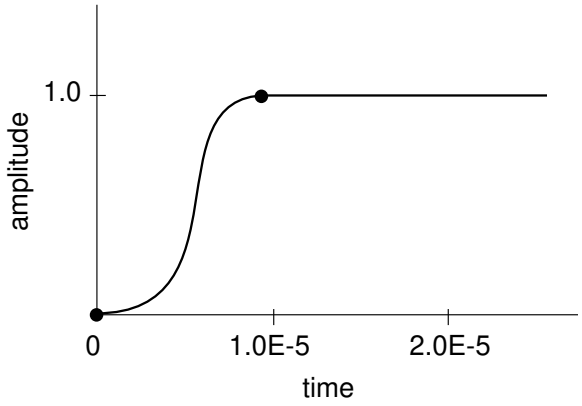


Figure 1: Amplitude definition using a smooth step amplitude curve.

Structural problems

In a static analysis the lowest mode of the structure usually dominates the response. Knowing the frequency and, correspondingly, the period of the lowest mode, you can estimate the time required to obtain the proper static response. To illustrate the problem of determining the proper loading rate, consider the deformation of a side intrusion beam in a car door by a rigid cylinder, as shown in *Figure 1*. The actual test is quasi-static.

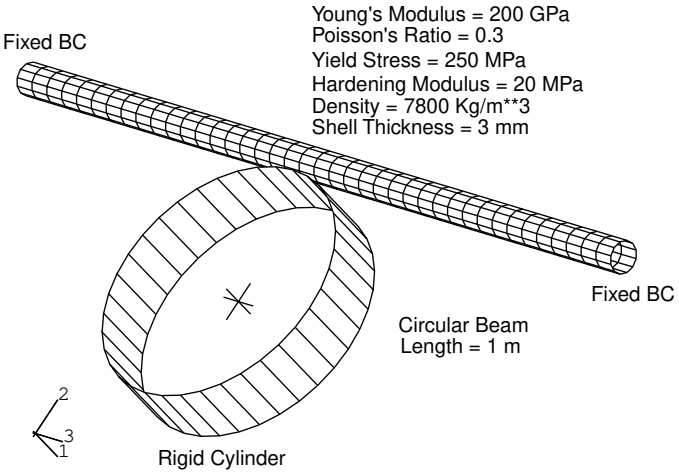


Figure 1: Rigid cylinder impacting beam.

The response of the beam varies greatly with the loading rate. At an extremely high impact velocity of 400 m/s, the deformation in the beam is highly localized, as shown in *Figure 2*. To obtain a better quasi-static solution, consider the lowest mode.

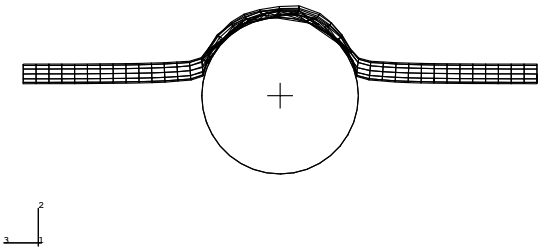


Figure 2: Impact velocity of 400 m/s.

The frequency of the lowest mode is approximately 250 Hz, which corresponds to a period of 4 milliseconds. The natural frequencies can be calculated easily using the eigenfrequency extraction procedure in Abaqus/Standard. To deform the beam by the desired 0.2 m in 4 milliseconds, the velocity of the cylinder is 50 m/s. While 50 m/s still seems like a high impact velocity, the inertial forces become secondary to the overall stiffness of the structure, and the deformed shape—shown in *Figure 3*—indicates a much better quasi-static response. While the overall structural response appears to be what we expect as a quasi-static solution, it is usually desirable to increase the loading time to 10 times the period of the lowest mode to be certain that the solution is truly quasi-static. To improve the results even further, the velocity of the rigid cylinder could be ramped up gradually—for example, using a smooth step amplitude curve—thereby easing the initial impact.

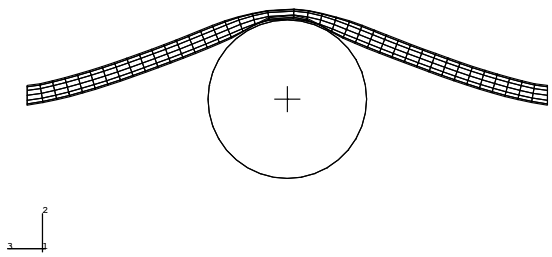


Figure 3: Impact velocity of 50 m/s.

Metal forming problems

Artificially increasing the speed of forming events is necessary to obtain an economical solution, but how large a speedup can we impose and still obtain an acceptable static solution? If the deformation of the sheet metal blank corresponds to the deformed shape of the lowest mode, the time period of the lowest structural mode can be used as a guideline for forming speed. However, in forming processes the rigid dies and punches can constrain the blank in such a way that its deformation may not relate to the structural modes. In such cases a general recommendation is to limit punch speeds to less than 1% of the sheet metal wave speed. For typical processes the punch speed is on the order of 1 m/s, while the wave speed of steel is approximately 5000 m/s. This recommendation, therefore, suggests a factor of 50 as an upper bound on the speedup of the punch velocity.

The suggested approach to determining an acceptable punch velocity involves running a series of analyses at various punch speeds in the range of 3 to 50 m/s. Perform the analyses in the order of fastest to slowest since the solution time is inversely proportional to the punch velocity. Examine the results of the analyses, and get a feel for how the deformed shapes, stresses, and strains vary with punch speed. Some indications of excessive punch speeds are unrealistic, localized stretching and thinning as well as the suppression of wrinkling. If you begin with a punch speed of, for example, 50 m/s, and decrease it from there, at some point the solutions will become similar from one punch speed to the next—an indication that the solutions are converging on a quasi-static solution. As inertial effects become less significant, differences in simulation results also become less significant.

As the loading rate is increased artificially, it becomes more and more important to apply the loads in a gradual and smooth manner. For example, the simplest way to load the punch is to impose a constant velocity throughout the forming step. Such a loading causes a sudden impact load onto the sheet metal blank at the start of the analysis, which propagates stress waves through the blank and may produce undesired results. The effect of any impact load on the results becomes more pronounced as the loading rate is increased. Ramping up the punch velocity from zero using a smooth step amplitude curve minimizes these adverse effects.

Springback

Springback is often an important part of a forming analysis because the springback analysis determines the shape of the final, unloaded part. While Abaqus/Explicit is well-suited for forming simulations, springback poses some special difficulties. The main problem with performing springback simulations within Abaqus/Explicit is the amount of time required to obtain a steady-state solution. Typically, the loads must be removed very carefully, and damping must be introduced to make the solution time reasonable. Fortunately, the close relationship between Abaqus/Explicit and Abaqus/Standard allows a much more efficient approach.

Since springback involves no contact and usually includes only mild nonlinearities, Abaqus/Standard can solve springback problems much faster than Abaqus/Explicit can. Therefore, the preferred approach to springback analyses is to import the completed forming model from Abaqus/Explicit into Abaqus/Standard.

Mass scaling

Mass scaling enables an analysis to be performed economically without artificially increasing the loading rate. Mass scaling is the only option for reducing the solution time in simulations involving a rate-dependent material or rate-dependent damping, such as dashpots. In such simulations increasing the loading rate is not an option because material strain rates increase by the same factor as the loading rate. When the properties of the model change with the strain rate, artificially increasing the loading rate artificially changes the process.

The following equations show how the stable time increment is related to the material density. As discussed in *Definition of the stability limit*, the stability limit for the model is the minimum stable time increment of all elements. It can be expressed as

$$\Delta t = \frac{L^e}{c_d},$$

where L^e is the characteristic element length and c_d is the dilatational wave speed of the material. The dilatational wave speed for a linear elastic material with Poisson's ratio equal to zero is given by

$$c_d = \sqrt{\frac{E}{\rho}},$$

where E is the Young's modulus and ρ is the material density.

According to the above equations, artificially increasing the material density, ρ, by a factor of f^2 decreases the wave speed by a factor of f and increases the stable time increment by a factor of f. Remember that when the global stability limit is increased, fewer increments are required to perform the same analysis, which is the goal of mass scaling. Scaling the mass, however, has exactly the same influence on inertial effects as artificially increasing the loading rate. Therefore, excessive mass scaling, just like excessive loading rates, can lead to erroneous solutions. The suggested approach to determining an acceptable mass scaling factor, then, is similar to the approach to determining an acceptable loading rate scaling factor. The only difference to the approach is that the speedup associated with mass scaling is the square root of the mass scaling factor, whereas the speedup associated with loading rate scaling is proportional to the loading rate scaling factor. For example, a mass scaling factor of 100 corresponds exactly to a loading rate scaling factor of 10.

There are several ways to implement mass scaling in your model using either fixed or variable mass scaling. The mass scaling definition can be changed from step to step, allowing great flexibility. Refer to *Mass Scaling* for details.

Energy balance

The most general means of evaluating whether or not a simulation is producing an appropriate quasi-static response involves studying the various model energies. The following is the energy balance equation in Abaqus/Explicit:

$$E_I + E_V + E_{FD} + E_{KE} + E_{IHE} - E_W - E_{PW} - E_{CW} - E_{MW} - E_{HF} = E_{total} = \text{constant},$$

where E_I is the internal energy, E_V is the viscous energy dissipated, E_{FD} is the frictional energy dissipated, E_{KE} is the kinetic energy, E_{IHE} is the internal heat energy, E_W is the work done by the externally applied loads, and E_{PW}, E_{CW}, and E_{MW} are the work done by contact penalties, by constraint penalties, and by propelling added mass, respectively. E_{HF} is the external heat energy through external fluxes. The sum of these energy components is E_{total}, which should be constant. In the numerical model E_{total} is only approximately constant, generally with an error of less than 1%.

To illustrate energy balance with a simple example, consider the uniaxial tensile test shown in *Figure 1*.

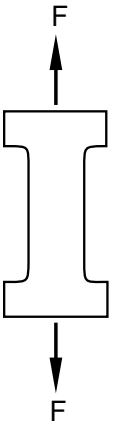


Figure 1: Uniaxial tensile test.

The energy history for the quasi-static test would appear as shown in *Figure 2*.

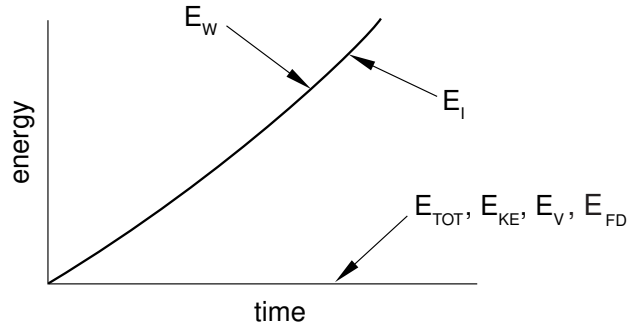


Figure 2: Energy history for quasi-static tensile test.

If a simulation is quasi-static, the work applied by the external forces is nearly equal to the internal energy of the system. The viscously dissipated energy is generally small unless viscoelastic materials, discrete dashpots, or material damping are used. We have already established that the inertial forces are negligible in a quasi-static analysis because the velocity of the material in the model is very small. The corollary to both of these conditions is that the kinetic energy is also small. As a general rule the kinetic energy of the deforming material should not exceed a small fraction (typically 5% to 10%) of its internal energy throughout most of the process.

When comparing the energies, remember that Abaqus/Explicit reports a global energy balance, which includes the kinetic energy of any rigid bodies with mass. Since only the deformable bodies are of interest when evaluating the results, the kinetic energy of the rigid bodies should be subtracted from E_{total} when evaluating the energy balance.

For example, if you are simulating a transport problem with rolling rigid dies, the kinetic energy of the rigid bodies may be a significant portion of the total kinetic energy of the model. In such cases you must subtract the kinetic energy associated with rigid body motions before a meaningful comparison with internal energy can be made.

Example: forming a channel in Abaqus/Explicit

In this example you will solve the channel forming problem using Abaqus/Explicit.
You will then compare the results from the Abaqus/Standard and Abaqus/Explicit analyses.

You will make modifications to the model created in *Contact* for the Abaqus/Standard analysis so that you are able to run it in Abaqus/Explicit. These modifications include adding density to the material model, changing the element library, and changing the steps. Before running the Abaqus/Explicit analysis, you will use the frequency extraction procedure in Abaqus/Standard to determine the time period required to obtain a proper quasi-static response.

In this section:

- *Preprocessing—rerunning the model with Abaqus/Explicit*
- *Forming analysis—attempt 2*
- *Discussion of the two forming attempts*
- *Methods of speeding up the analysis*
- *Springback analysis in Abaqus/Standard*

Preprocessing—rerunning the model with Abaqus/Explicit

Use Abaqus/CAE to modify the model for this simulation.
Abaqus provides scripts that replicate the complete analysis model for this problem. Run one of these scripts if you encounter difficulties following the instructions given below or if you wish to check your work. Scripts are available in the following locations:

- A Python script for this example is provided in *Forming a channel*. Instructions on how to fetch the script and run it within Abaqus/CAE are given in *Example Files*.
- A plug-in script for this example is available in the Abaqus/CAE Plug-in toolset. To run the script from Abaqus/CAE, select **Plug-ins->Abaqus->Getting Started**; highlight **Forming a channel**; and click **Run**. For more information about the Getting Started plug-ins, see *Running the Getting Started with Abaqus examples*.

Before starting, open the model database file for the channel forming example created in *Abaqus/Standard 2D example: forming a channel*.

In this section:

- *Determining an appropriate step time*
- *Abaqus/Explicit forming analysis*
- *Strategy for evaluating the results*
- *Evaluating the results*

Determining an appropriate step time

You can determine an appropriate step time.

Loading rates discusses the procedures for determining the appropriate step time for a quasi-static process. We can determine an approximate lower bound on step time duration if we know the lowest natural frequency, the *fundamental* frequency, of the blank. One way to obtain such information is to run a frequency analysis in Abaqus/Standard. In this forming analysis the punch deforms the blank into a shape similar to the lowest mode. Therefore, it is important that the time for the forming stage is greater than or equal to the time period for the lowest mode if you wish to model structural, as opposed to localized, deformation.

1. Copy the existing model to a model named `Frequency`. Make all of the following changes to the `Frequency` model. In the frequency extraction analysis you will replace all existing steps with a single frequency extraction step. In addition, you will delete all of the rigid body tools and contact interactions; they are not necessary for determining the fundamental frequency of the blank.
2. Add a density of `7800` to the material model `Steel`.
3. Delete the die, holder, and punch part instances. These rigid parts are not necessary for the frequency analysis.

   Tip: You can delete any part instance using the Model Tree by expanding **Instances** underneath the **Assembly** container, clicking mouse button 3 on the instance name, and selecting **Delete** from the menu that appears.

4. Replace the existing steps with a single frequency extraction step.
   a. Delete the step `Move punch`.
   b. In the Model Tree, click mouse button 3 on the step `Holder force` and select **Replace** from the menu that appears.
   c. In the **Replace Step** dialog box, select **Frequency** from the list of available **Linear perturbation** procedures. Enter the step description `Frequency modes`; select the Lanczos eigensolver option; and request five eigenvalues. Rename the step `Extract Frequencies`.
   d. Suppress the **DOF Monitor**.

   Note:

   Since the frequency extraction step is a linear perturbation procedure, nonlinear material properties will be ignored. In this analysis the left end of the blank is constrained in the *x*-direction and cannot rotate about the normal; however, it is not constrained in the *y*-direction. Therefore, the first mode extracted will be a rigid body mode. The frequency of the second mode will determine the appropriate time period for the quasi-static analysis in Abaqus/Explicit.

5. Delete all contact interactions.
6. Open the **Boundary Condition Manager**, and examine the boundary conditions in the `Extract Frequencies` step. Delete all boundary conditions except the boundary condition named `CenterBC`. This leaves the blank constrained with a symmetry boundary condition applied to the left end.
7. Remesh the blank if necessary.
8. Create a job named `Forming-Frequency` with the following job description: `Channel forming -- frequency analysis`. Submit the job for analysis, and monitor the solution progress.
9. When the analysis is complete, enter the Visualization module and open the output database file created by this job. From the main menu bar, select **Plot->Deformed Shape**; or use the  tool in the toolbox.

The deformed model shape for the first vibration mode is plotted (it is a rigid body mode). Advance the plot to the second mode of the blank. Superimpose the undeformed model shape on the deformed model shape.

The frequency analysis shows that the blank has a fundamental frequency of 140 Hz, corresponding to a period of 0.00714 s. *Figure 1* shows the displaced shape of the second mode. We now know that the shortest step time for the forming analysis is 0.00714 s.

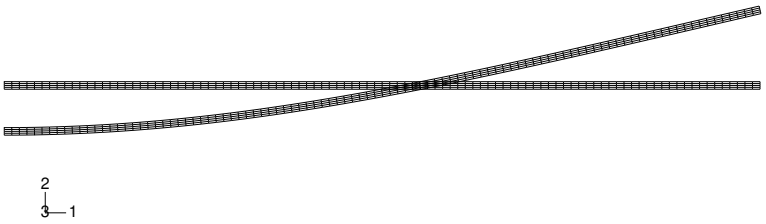


Figure 1: Second mode of the blank from the Abaqus/Standard frequency analysis.

Abaqus/Explicit forming analysis

The goal of the forming process is to quasi-statically form a channel with a punch displacement of 0.03 m. In selecting loading rates for quasi-static analyses, it is recommended that you begin with faster loading rates and decrease the loading rates as necessary to converge on a quasi-static solution more quickly. However, if you wish to increase the likelihood of a quasi-static result in your first analysis attempt, you should consider step times that are a factor of 10 to 50 times slower than that corresponding to the fundamental frequency. In this analysis you will start with a time period of 0.007 s for the forming analysis step, which is based on the frequency analysis performed in Abaqus/Standard, which shows that the blank has a fundamental frequency of 140 Hz, corresponding to a time period of 0.00714 s. This time period corresponds to a constant punch velocity of 4.3 m/s. You will examine the kinetic and internal energy results carefully to check that the solution does not include significant dynamic effects.

Creating the Abaqus/Explicit forming analysis

Copy the `Standard` model to a model named `Explicit`. Make all subsequent model changes to the `Explicit` model. To begin, edit the `Steel` material definition to include a mass density of 7800 kg/m^3.

A concentrated force will be applied to the blank holder. To compute the dynamic response of the holder, a point mass must be assigned to its rigid body reference point. The actual mass of the holder is not important; what is important is that the mass should be of the same order of magnitude as the mass of the blank (0.78 kg) to minimize noise in the contact calculations. Choose a point mass value of 0.1 kg. To assign the mass, expand **Engineering Features** underneath the **Holder** in the **Parts** container in the Model Tree. In the list that appears, double-click **Inertias**. In the **Create Inertia** dialog box that appears, enter the name `PointMass` and click **Continue**. Select the reference point of the holder, and assign it a mass of `0.1` kg.

For the first attempt of this metal forming analysis, you will use tabular amplitude curves with the default smoothing parameter for both the application of the holder force and the punch stroke. Create a tabular amplitude curve for application of the holder force named `Ramp1` using the data in *Table 1*. Define a second tabular amplitude curve for the punch stroke named `Ramp2` using the data in *Table 2*.

Table 1: Ramp amplitude data for `Ramp1` and `Smooth1`.

Time (sec)	Amplitude
0.0	0.0
0.0001	1.0

Table 2: Ramp amplitude data for `Ramp2` and `Smooth2`.

Time (sec)	Amplitude
0.0	0.0
0.007	1.0

As with the Abaqus/Standard analysis, you will need two steps for the Abaqus/Explicit analysis. In the first step the holder force is applied; in the second step the punch stroke is applied. Delete the step named `Move punch`. Replace the step named `Holder force` with an explicit dynamics step, and specify a time period of `0.0001` s. This time period is appropriate for the application of the holder force because it is long enough to avoid dynamic effects but short enough to prevent a significant impact on the run time for the job. Enter `Apply holder force` as the step description. Create a second explicit dynamics step named `Move punch` with a time period of `0.007` s. Enter `Apply punch stroke` as the step description.

To help determine how closely the analysis approximates the quasi-static assumption, the various energy histories will be useful. Especially useful is comparing the kinetic energy to the internal strain energy. The energy history is written to the output database file by default. Modify the history output request for the punch reference node to request output at 200 evenly spaced intervals.

Open the **Load Manager**, and modify the concentrated force named `RefHolderForce` in the step named `Holder force` so that the amplitude definition for this load is set to `Ramp1`.

Change the displacement boundary condition `RefPunchBC` in the `Move punch` step so that **U2** is set to –0.03 m in the step. Use the amplitude curve `Ramp2` for this boundary condition.

Monitoring the value of a degree of freedom

In this model you will monitor the vertical displacement (degree of freedom 2) of the punch's reference node throughout the step. Because the **DOF Monitor** was set to monitor the vertical displacement of `RefPunch` in the Abaqus/Standard forming analysis, you do not need to make any changes.

Mesh creation and job definition

Change the family of the elements used to mesh the blank to **Explicit**, and specify enhanced hourglass control.

Create a job named `Forming-1`. Give the job the following description: `Channel forming -- attempt 1`.

Before you run the forming analysis, you may wish to know how many increments the analysis will take and, consequently, how much computer time the analysis requires. You can use the **Verify Mesh**  tool to obtain an estimate of the stable time increment (the option is located in the **Size Metrics** tabbed page of the **Verify Mesh** dialog box). Knowing the stable time increment, which in this case does not change much from increment to increment, you can determine how many increments are required to complete the forming stage. Once the analysis begins, you can get an idea of how much CPU time is required per increment and, consequently, how much CPU time the analysis requires.

The stable time increment for this analysis is approximately 3.8×10^{-8} s. Therefore, the forming stage requires approximately 185,000 increments for a step time of 0.007 s.

Save your model to a model database file, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate the cause of any warning messages.

Once the analysis is underway, an *X–Y* plot of the values of the degree of freedom that you selected to monitor (the punch's vertical displacement) appears in a separate viewport. From the main menu bar, select **Viewport**->**DOF Monitor: Forming-1** to follow the progression of the punch's displacement in the 2-direction over time as the analysis runs.

Strategy for evaluating the results

Before looking at the results that are ultimately of interest, such as stresses and deformed shapes, we need to determine whether or not the solution is quasi-static. One good approach is to compare the kinetic energy history to the internal energy history. In a metal forming analysis most of the internal energy is due to plastic deformation. In this model the blank is the primary source of kinetic energy (the motion of the holder is negligible, and the punch and die have no mass associated with them). To determine whether an acceptable quasi-static solution has been obtained, the kinetic energy of the blank should be no greater than a few percent of its internal energy. For greater accuracy, especially when springback stresses are of interest, the kinetic energy should be lower. This approach is very useful because it applies to all types of metal forming processes and does not require any intuitive understanding of the stresses in the model; many forming processes may be too complex to permit an intuitive feel for the results.

While a good primary indication of the caliber of a quasi-static analysis, the ratio of kinetic energy to internal energy alone is not adequate to confirm the quality. You should also evaluate the two energies independently to determine whether they are reasonable. This part of the evaluation takes on increased importance when accurate springback stress results are needed because an accurate springback stress solution is highly dependent on accurate plasticity results. Even if the kinetic energy is fairly small, if it contains large oscillations, the model could be experiencing significant plasticity. Generally, we expect smooth loading to produce smooth results; if the loading is smooth but the energy results are oscillatory, the results may be inadequate. Since an energy ratio is incapable of showing such behavior, you should also study the kinetic energy history itself to see whether it is smooth or noisy.

If the kinetic energy does not indicate quasi-static behavior, it can be useful to look at velocity histories at some nodes to get an understanding of the model's behavior in various regions. Such velocity histories can indicate which regions of the model are oscillating and causing the high kinetic energies.

Evaluating the results

Enter the Visualization module, and open the output database created by this job (`Forming-1.odb`). Plot the whole model kinetic (`ALLKE`) and internal (`ALLIE`) energies.

History plots of the kinetic and internal energies for the whole model appear as shown in *Figure 1* and *Figure 2*, respectively.

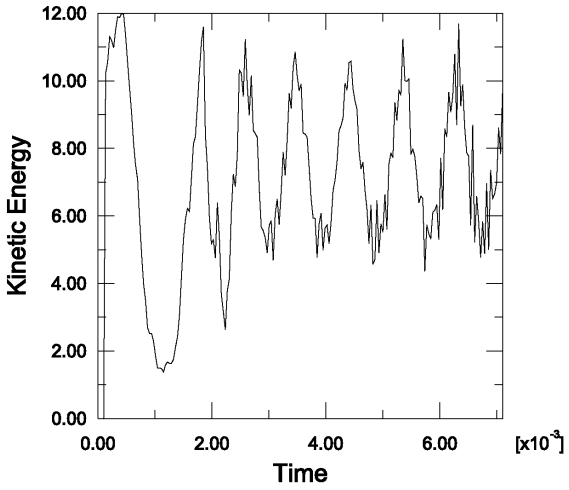


Figure 1: Kinetic energy history for forming analysis, attempt 1.

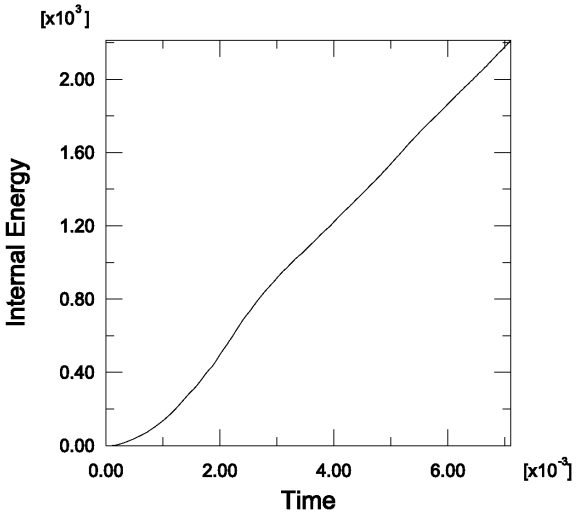


Figure 2: Internal energy history for forming analysis, attempt 1.

The kinetic energy history shown in *Figure 1* oscillates significantly. In addition, the kinetic energy history has no clear relation to the forming of the blank, which indicates the inadequacy of this analysis. In this analysis the punch velocity remains constant, while the kinetic energy—which is primarily due to the motion of the blank—is far from constant.

Comparing *Figure 1* and *Figure 2* shows that the kinetic energy is a small fraction (less than 1%) of the internal energy through all but the very beginning of the analysis. The criterion that kinetic energy must be small relative to internal energy has been satisfied, even for this severe loading case.

Although the kinetic energy of the model is a small fraction of the internal energy, it is still quite noisy. Therefore, we should change the simulation in some way to obtain a smoother response.

Forming analysis—attempt 2

Even if the punch actually moves at a nearly constant velocity, the results of the first simulation attempt indicate it is desirable to use a different amplitude curve that allows the blank to accelerate more smoothly. When considering what type of loading amplitude to use, remember that smoothness is important in all aspects of a quasi-static analysis. The preferred approach is to move the punch as smoothly as possible the desired distance in the desired amount of time.

We will now analyze the forming stage using a smoothly applied holder force and a smoothly applied punch displacement; we will compare the results to those obtained earlier. Refer to *Smooth amplitude curves* for an explanation of the smooth step amplitude curve.

Define a smooth step amplitude curve named `Smooth1`. Enter the amplitude data given in *Table 1*. Create a second smooth step amplitude curve named `Smooth2` using the amplitude data given in *Table 2*. Modify the `RefHolderForce` load in the `Holder force` step so that it refers to the `Smooth1` amplitude. Modify the displacement boundary condition `RefPunchBC` in the `Move punch` step so that it refers to the `Smooth2` amplitude. By specifying an amplitude of 0.0 at the beginning of the step and an amplitude of 1.0 at the end of the step, Abaqus/Explicit creates an amplitude definition that is smooth in both its first and second derivatives. Therefore, using a smooth step amplitude curve for the displacement control also assures us that the velocity and acceleration are smooth.

Create a job named `Forming-2`. Give the job the following description: `Channel forming -- attempt 2`.

Save your model, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate the cause of any warning messages.

Evaluating the results for attempt 2

The kinetic energy history is shown in *Figure 1*. The response of the kinetic energy is clearly related to the forming of the blank: the value of kinetic energy peaks in the middle of the second step, corresponding to the time when the punch velocity is the greatest. Thus, the kinetic energy is appropriate and reasonable.

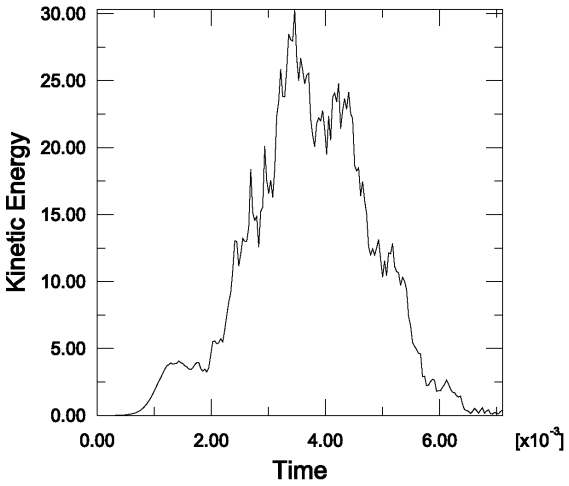


Figure 1: Kinetic energy history for forming analysis, attempt 2.

The internal energy for attempt 2, shown in *Figure 2*, shows a smooth increase from zero up to the final value. Again, the ratio of kinetic energy to internal energy is quite small and appears to be acceptable. *Figure 3* compares the internal energy in the two forming attempts.

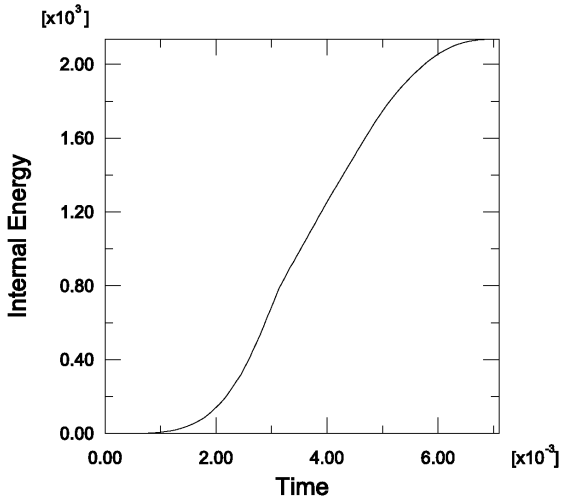


Figure 2: Internal energy history for forming analysis, attempt 2.

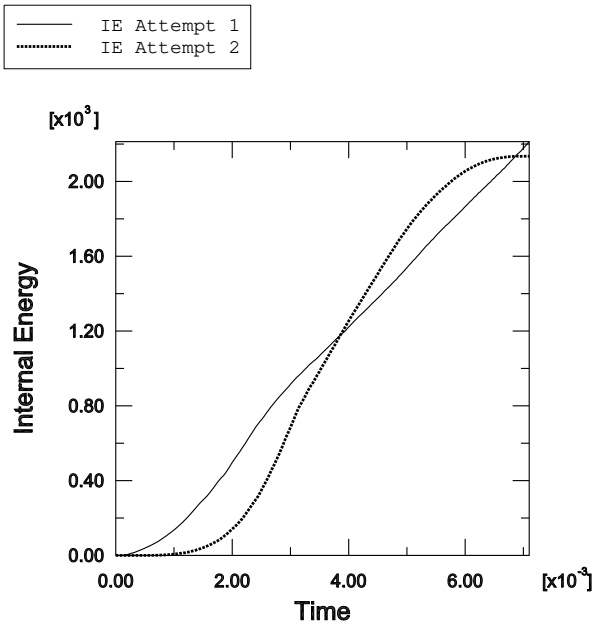


Figure 3: Comparison of internal energies for the two attempts of the forming analysis.

Discussion of the two forming attempts

Our initial criteria for evaluating the acceptability of the results was that the kinetic energy should be small compared to the internal energy. What we found was that even for the most severe case, attempt 1, this condition seems to have been met adequately. The addition of a smooth step amplitude curve helped reduce the oscillations in the kinetic energy, yielding a satisfactory quasi-static response.

The additional requirements—that the histories of kinetic energy and internal energy must be appropriate and reasonable—are very useful and necessary, but they also increase the subjectivity of evaluating the results. Enforcing these requirements in general for more complex forming processes may be difficult because these requirements demand some intuition regarding the behavior of the forming process.

Results of the forming analysis

Now that we are satisfied that the quasi-static solution for the forming analysis is adequate, we can study some of the other results of interest. *Figure 1* shows a comparison of the Mises stress in the blank obtained with Abaqus/Standard and Abaqus/Explicit.

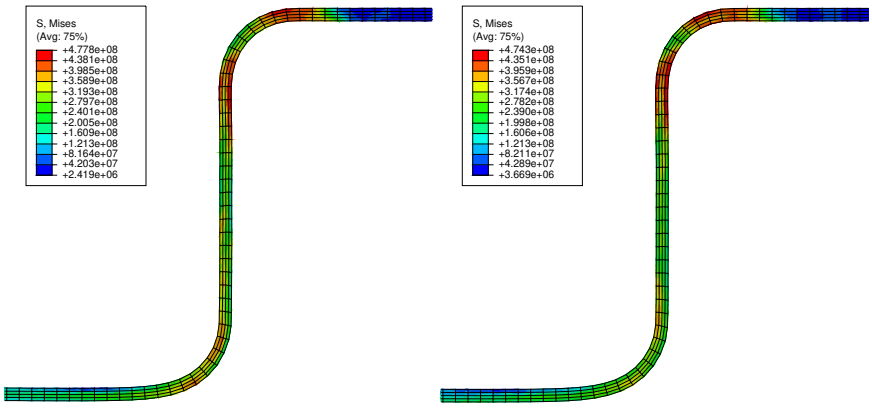


Figure 1: Contour plot of Mises stress in Abaqus/Standard (left) and Abaqus/Explicit (right) channel forming analyses.

The plot shows that the peak stresses in the Abaqus/Standard and Abaqus/Explicit analyses are within 1% of each other and that the overall stress contours of the blank are very similar. To further examine the validity of the quasi-static analysis results, you should compare the equivalent plastic strain results and final deformed shapes from the two analyses.

Figure 2 shows contour plots of the equivalent plastic strain in the blank, and *Figure 3* shows an overlay plot of the final deformed shape predicted by the two analyses.

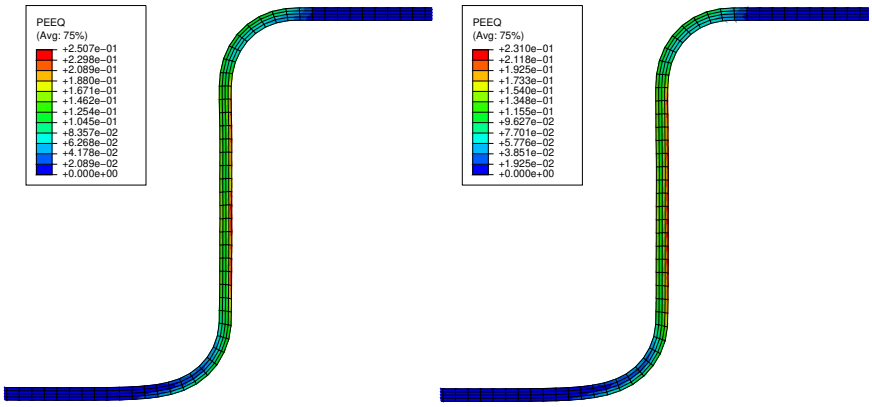


Figure 2: Contour plot of PEEQ in Abaqus/Standard (left) and Abaqus/Explicit (right) channel forming analyses.

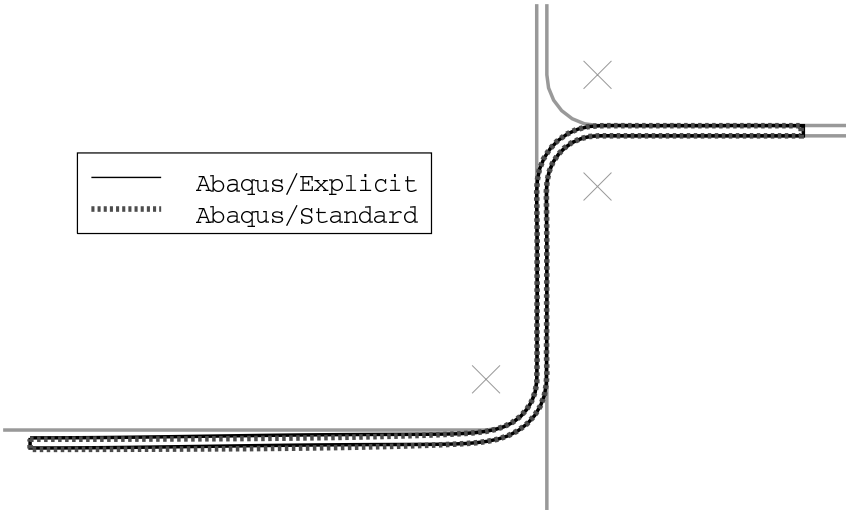


Figure 3: Final deformed shape in Abaqus/Standard and Abaqus/Explicit forming analyses.

The equivalent plastic strain results for the Abaqus/Standard and Abaqus/Explicit analyses are within 5% of each other. In addition, the final deformed shape comparison shows that the explicit quasi-static analysis results are in excellent agreement with the results from the Abaqus/Standard static analysis.

You should also compare the steady punch force predicted by the Abaqus/Standard and Abaqus/Explicit analyses.

To compare the punch force-displacement histories:

1. Save the punch displacement (`U2`) and reaction force (`RF2`) history data from the Abaqus/Standard analysis as `U2-std` and `RF2-std`, respectively.
2. Similarly, save punch displacement (`U2`) and reaction force (`RF2`) history data from the Abaqus/Explicit analysis as `U2-xpl` and `RF2-xpl`, respectively.

   Next, you will operate on saved *X*–*Y* data to create the force-displacement curves. In the force-displacement plot we would like the downward motion of the punch to be represented as a positive value; therefore, when

you create the force-displacement curves include a negative sign before the displacement history data so that motion in the negative 2-direction will be positive.

3. In the Results Tree, double-click **XYData**; then select **Operate on XY data** in the **Create XY Data** dialog box. Click **Continue**.
4. In the **Operate on XY Data** dialog box, combine the force and displacement history data from the Abaqus/Standard analysis to create a force-displacement curve. The expression at the top of the dialog box should appear as:

   ```
   combine ( -"U2-std", "RF2-std" )
   ```

5. Click **Save As** to save the calculated displacement curve as `forceDisp-std`.
6. In the **Operate on XY Data** dialog box, combine the force and displacement history data from the Abaqus/Explicit analysis to create a force-displacement curve. The expression at the top of the dialog box should appear as:

   ```
   combine ( -"U2-xpl", "RF2-xpl" )
   ```

7. Click **Save As** to save the calculated displacement curve as `forceDisp-xpl`.
8. Plot `forceDisp-std` and `forceDisp-xpl` in the viewport.

   There is significantly more noise in the Abaqus/Explicit results compared to the Abaqus/Standard results because Abaqus/Explicit simulates a quasi-static response while Abaqus/Standard solves for true static equilibrium. You will use an Abaqus/CAE *X*–*Y* data filter to remove more of the solution noise from the Abaqus/Explicit force-displacement curve. The Abaqus/CAE *X*–*Y* data filters should only be applied to *X*–*Y* data whose *X*-value is time. This avoids confusion regarding the meaning of the filter cutoff frequency and prevents problems with the data regularization that is performed internally before the filter is applied. Consequently, you will not filter `forceDisp-xpl` directly, but rather you will filter `U2-xpl` and `RF2-xpl` individually before combining them to create a new force-displacement curve. It is best to apply the same filter operations (both during the analysis and during postprocessing) to any two *X*–*Y* data objects that will be combined. This will ensure that any distortions due to filtering (such as time delays) are uniformly applied to the combined data.
9. In the **Operate on XY Data** dialog box, filter the force history data using a Butterworth filter with a cutoff frequency of 1100 Hz. The expression at the top of the dialog box should appear as:

   ```
   butterworthFilter(xyData="RF2-xpl",cutoffFrequency=1100)
   ```

   Note:

   Choosing an appropriate filter cutoff frequency takes engineering judgment and a good understanding of the physical system being modeled. Often an iterative approach (beginning with a relatively high cutoff frequency and then gradually reducing it) can be used to find a cutoff frequency that removes solution noise with minimal distortion of the underlying physical solution. Knowledge of the system's natural frequencies can also assist in the determination of appropriate filter cutoff frequencies. For this example, we performed a frequency extraction analysis to determine the fundamental frequency of the undeformed blank (140 Hz); however, the blank at the end of the forming step will have a fundamental frequency that is considerably higher. If you perform a natural frequency extraction analysis on the final model configuration, you will find that the fundamental frequency at the end of the forming step is approximately 1000 Hz. Hence, a cutoff frequency that is slightly larger than this value is a good choice for this model.

10. Click **Save As** to save the calculated displacement curve as `RF2-xpl-bw1100`.
11. Similarly, filter the displacement history data using a Butterworth filter with a cutoff frequency of 1100 Hz. The expression at the top of the **Operate on XY Data** dialog box should appear as:

    ```
    butterworthFilter(xyData="U2-xpl",cutoffFrequency=1100)
    ```

12. Click **Save As** to save the calculated displacement curve as `U2-xpl-bw1100`.
13. Combine the filtered Abaqus/Explicit force and displacement histories. The expression at the top of the **Operate on XY Data** dialog box should appear as:

    ```
    combine ( -"U2-xpl-bw1100", "RF2-xpl-bw1100" )
    ```

14. Click **Save As** to save the calculated displacement curve as `forceDisp-xpl-bw1100`.
15. Add `forceDisp-xpl-bw1100` to the plot of `forceDisp-std` and `forceDisp-xpl`. Customize the plot appearance to obtain a plot similar to *Figure 4*.

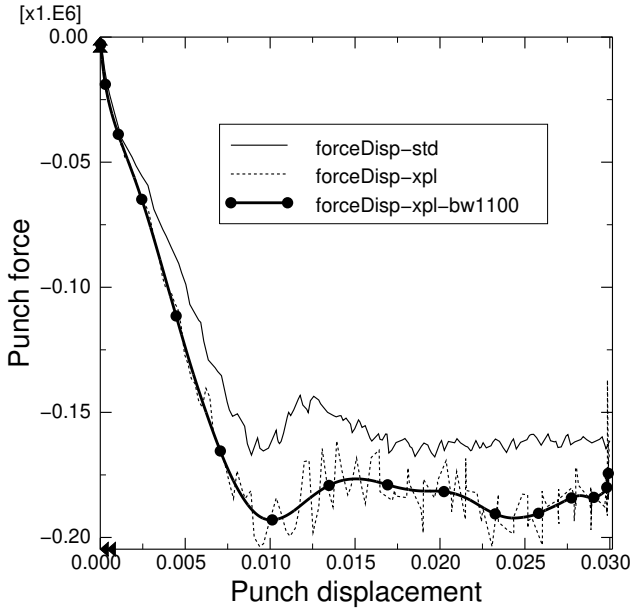


Figure 4: Steady punch force comparison for Abaqus/Standard and Abaqus/Explicit.

As seen in *Figure 4*, the steady punch force predicted by Abaqus/Explicit is approximately 12% higher than that predicted by Abaqus/Standard. The differences between the Abaqus/Standard and Abaqus/Explicit results are primarily due to two factors. First, Abaqus/Explicit regularizes the material data. Second, friction effects are handled slightly differently in the two analysis products; Abaqus/Standard uses penalty friction, whereas Abaqus/Explicit uses kinematic friction.

From these comparisons it is clear that both Abaqus/Standard and Abaqus/Explicit are capable of handling difficult contact analyses such as this one. However, there are some advantages to running this type of analysis in Abaqus/Explicit: Abaqus/Explicit is able to handle complex contact conditions more readily. However, when choosing Abaqus/Explicit for quasi-static analysis, you should be aware that you may need to iterate on an appropriate loading rate. In determining the loading rate, it is recommended that you begin with faster loading rates and decrease the loading rate as necessary. This will help optimize the run time for the analysis.

Methods of speeding up the analysis

Now that we have obtained an acceptable solution to the forming analysis, we can try to obtain similar acceptable results using less computer time.
Most forming analyses require too much computer time to be run in their physical time scale because the actual time period of forming events is large by explicit dynamics standards; running in an acceptable amount of computer time often requires making changes to the analysis to reduce the computer cost.

There are two ways to reduce the cost of the analysis:

1. Artificially increase the punch velocity so that the same forming process occurs in a shorter step time. This method is called *load rate scaling*.
2. Artificially increase the mass density of the elements so that the stability limit increases, allowing the analysis to take fewer increments. This method is called *mass scaling*.

Unless the model has rate-dependent materials or damping, these two methods effectively do the same thing.

Determining acceptable mass scaling

Loading rates and *Metal forming problems* discuss how to determine acceptable scaling of the loading rate or mass to reduce the run time of a quasi-static analysis. The goal is to model the process in the shortest time period in which inertial forces remain insignificant. There are bounds on how much scaling can be used while still obtaining a meaningful quasi-static solution.

As discussed in *Loading rates*, we can use the same methods to determine an appropriate mass scaling factor as we would use to determine an appropriate load rate scaling factor. The difference between the two methods is that a load rate scaling factor of f has the same effect as a mass scaling factor of f^2. Originally, we assumed that a step time on the order of the period of the fundamental frequency of the blank would be adequate to produce quasi-static results. By studying the model energies and other results, we were satisfied that these results were acceptable. This technique produced a punch velocity of approximately 4.3 m/s. Now we will accelerate the solution time using mass scaling and compare the results against our unscaled solution to determine whether the scaled results are acceptable. We assume that scaling can only diminish, not improve, the quality of the results. The objective is to use scaling to decrease the computer time and still produce acceptable results.

Our goal is to determine what scaling values still produce acceptable results and at what point the scaled results become unacceptable. To see the effects of both acceptable and unacceptable scaling factors, we investigate a range of scaling factors on the stable time increment size from $\sqrt{5}$ to 5; specifically, we choose $\sqrt{5}$, $\sqrt{10}$, and 5. These speedup factors translate into mass scaling factors of 5, 10, and 25, respectively.

To apply a mass scaling factor:

1. Create a set containing the blank named `Blank`.
2. Edit the step `Holder force`.
3. In the **Edit Step** dialog box, click the **Mass scaling** tab and toggle on **Use scaling definitions below**.
4. Click **Create**. Accept the default selection of semi-automatic mass scaling. Select set **Blank** as the region of application, and enter a value of `5` as the scale factor.

Create a job named `Forming-3--sqrt5`. Give the job the description `Channel forming -- attempt 3, mass scale factor=5`.

Save your model, and submit the job for analysis. Monitor the solution progress; correct any modeling errors that are detected, and investigate the cause of any warning messages.

When the job is finished, change the mass scaling factor to 10. Create and run a new job named `Forming-4--sqrt10`. When this job has completed, change the mass scaling factor again to 25; create and run a new job named `Forming-5--5`. For each of the last two jobs, modify the job descriptions as appropriate.

First, we will look at the effect of mass scaling on the equivalent plastic strains and the displaced shape. We will then see whether the energy histories provide a general indication of the analysis quality.

Evaluating the results with mass scaling

One of the results of interest in this analysis is the equivalent plastic strain, PEEQ. Since we have already seen the contour plot of PEEQ at the completion of the unscaled analysis in *Figure 2*, we can compare the results from each of the scaled analyses with the unscaled analysis results. *Figure 1* shows PEEQ for a speedup of $\sqrt{5}$ (mass scaling of 5), *Figure 2* shows PEEQ for a speedup of $\sqrt{10}$ (mass scaling of 10), and *Figure 3* shows PEEQ for a speedup of 5 (mass scaling of 25). *Figure 4* compares the internal and kinetic energy histories for each case of mass scaling. The mass scaling case using a factor of 5 yields results that are not significantly affected by the increased loading rate. The case with a mass scaling factor of 10 shows a high kinetic-to-internal energy ratio, yet the results seem reasonable when compared to those obtained with slower loading rates. Thus, this is likely close to the limit on how much this analysis can be sped up. The final case, with a mass scaling factor of 25, shows evidence of strong dynamic effects: the kinetic-to-internal energy ratio is quite high, and a comparison of the final deformed shapes among the three cases demonstrates that the deformed shape is significantly affected in the last case.

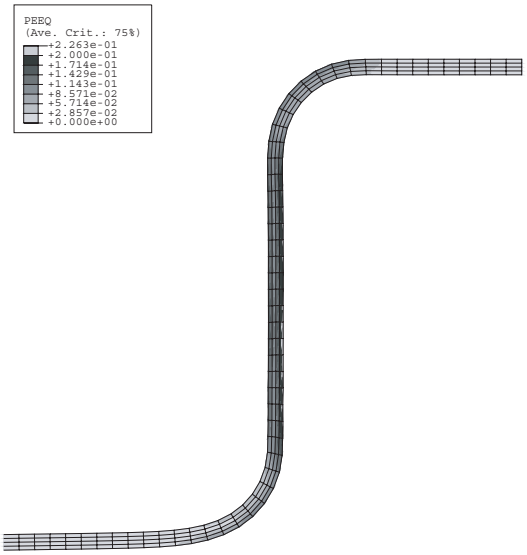


Figure 1: Equivalent plastic strain PEEQ for speedup of $\sqrt{5}$ (mass scaling of 5).

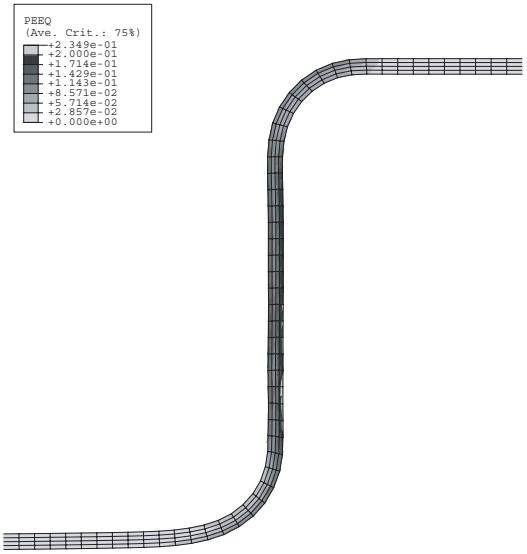


Figure 2: Equivalent plastic strain PEEQ for speedup of $\sqrt{10}$ (mass scaling of 10).

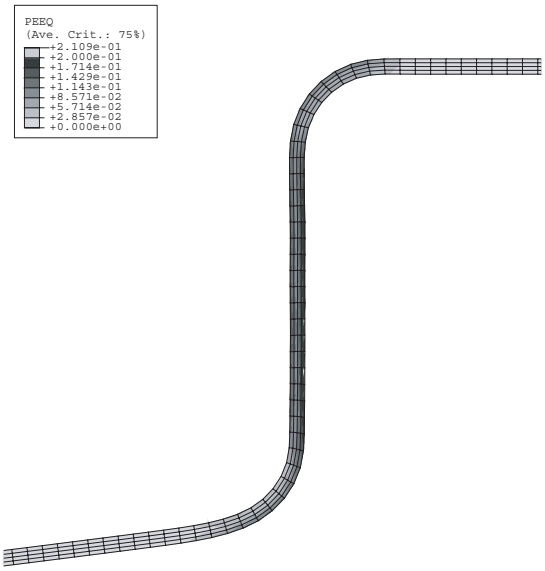


Figure 3: Equivalent plastic strain PEEQ for speedup of 5 (mass scaling of 25).

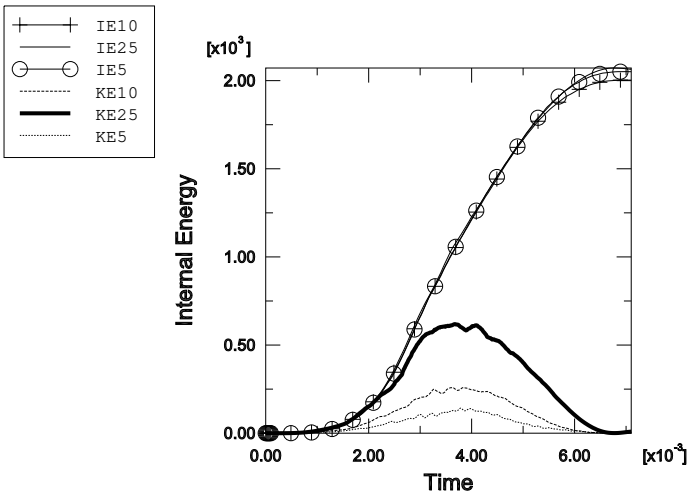


Figure 4: Kinetic and internal energy histories for mass scaling factors of 5, 10, and 25, corresponding to speedup factors of $\sqrt{5}$, $\sqrt{10}$, and 5, respectively.

Discussion of speedup methods

As the mass scaling increases, the solution time decreases. The quality of the results also decreases because dynamic effects become more prominent, but there is usually some level of scaling that improves the solution time without sacrificing the quality of the results. Clearly, a speedup of 5 is too great to produce quasi-static results for this analysis.

A smaller speedup, such as $\sqrt{5}$, does not alter the results significantly. These results would be adequate for most applications, including springback analyses. With a scaling factor of 10 the quality of the results begins to diminish, while the general magnitudes and trends of the results remain intact. Correspondingly, the ratio of kinetic energy to internal energy increases noticeably. The results for this case would be adequate for many applications but not for accurate springback analysis.

Springback analysis in Abaqus/Standard

While it is possible to perform springback analyses within Abaqus/Explicit, Abaqus/Standard is much more efficient at solving springback analyses.
Since springback analyses are simply static simulations without external loading or contact, Abaqus/Standard can obtain a springback solution in just a few increments. Conversely, Abaqus/Explicit must obtain a dynamic solution over a time period that is long enough for the solution to reach a steady state. For efficiency Abaqus has the capability to transfer results back and forth between Abaqus/Explicit and Abaqus/Standard, allowing you to perform forming analyses in Abaqus/Explicit and springback analyses in Abaqus/Standard.

You will create a new model that imports the results from the analysis with a speedup of $\sqrt{5}$ (mass scaling of 5) and perform a springback analysis. Thus, copy the `Explicit` model to a model named `Import`. Make all subsequent model changes to the `Import` model.

Since only the blank needs to be imported, begin by deleting the following features from the `Import` model:

- Part instances **Punch-1**, **Holder-1**, and **Die-1**.
- Sets **RefDie**, **RefHolder**, and **RefPunch**.
- All surfaces.
- All contact interactions and properties.
- Both analysis steps and all boundary conditions.

Next, create a general static step named `springback`. Set the initial time increment to `0.1`, and include the effects of geometric nonlinearity (note that the Abaqus/Explicit analysis considered them; this is the default setting in Abaqus/Explicit). Springback analyses can suffer from instabilities that adversely affect convergence. Thus, include automatic stabilization to prevent this problem. Use the default value for the dissipated energy fraction. Toggle off adaptive stabilization.

Next, define the initial state for the springback model based on the final state of the forming model.

1. In the Model Tree, double-click the **Predefined Fields** container. In the **Create Predefined Field** dialog box, select **Initial** as the step, **Other** as the category, and **Initial state** as the type. Click **Continue**.
2. In the viewport, select the blank as the instance to which the initial state will be assigned and click **Done** in the prompt area.
3. In the **Edit Predefined Field** dialog box that appears, enter the job name `Forming-3--sqrt5`.

   This corresponds to the job with a speedup of $\sqrt{5}$. Accept all other default settings, and click **OK**.

   This will cause the state of the model—stresses, strains, etc.—to be imported. By not updating the reference configuration, the springback displacements will be referred to the original undeformed configuration. This will allow for continuity in the displacements in the event additional forming stages are required.

You must redefine the boundary conditions, which are not imported. Impose the same XSYMM-type displacement boundary conditions that were imposed in the Abaqus/Explicit model on the set **Center**.

To remove rigid body motion, it is necessary to fix a single point in the blank, such as the node at the center of the left edge, in the 2-direction (in this way you impose no unnecessary constraints). Rather than apply a displacement boundary condition to this point, apply a zero-velocity boundary condition to fix this node at its position at the end of the forming stage (click the  tool to display the mesh in the Load module; create a *node*-based set at this node and then apply the appropriate boundary condition). This will allow the model to retain continuity in the blank location through any additional forming stages that may follow.

Create a new job named `springback`, and submit it for analysis.

Results of the springback analysis

Figure 1 overlays (**View->Overlay Plot**) the deformed shape of the blank after the forming and springback stages (the forming stage corresponds to the last frame of the Abaqus/Explicit output database file, while the springback stage corresponds to the final frame of the Abaqus/Standard output database file). The springback result is necessarily dependent on the accuracy of the forming stage preceding it. In fact, springback results are highly sensitive to errors in the forming stage, more sensitive than the results of the forming stage itself.

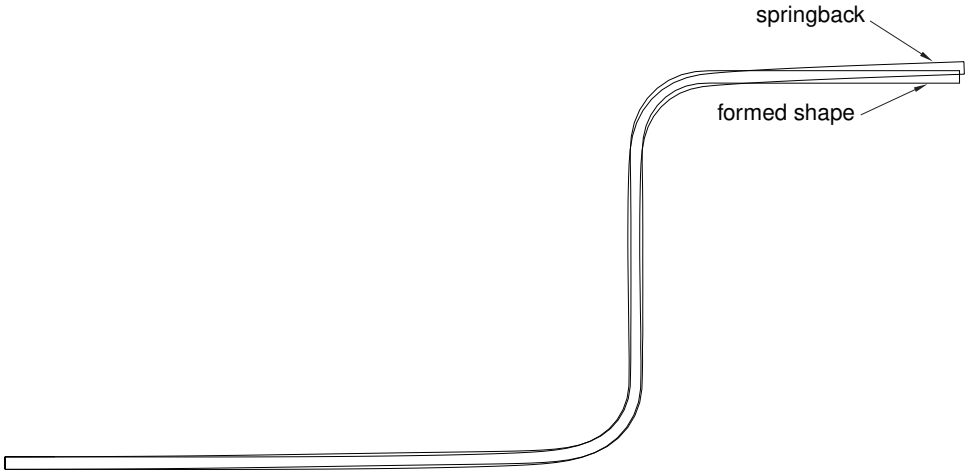


Figure 1: Deformed model shapes following forming and springback.

You should also plot the blank's internal energy ALLIE and compare it with the static stabilization energy ALLSD that is dissipated. The stabilization energy should be a small fraction of the internal energy to have confidence in the results. *Figure 2* shows a plot of these two energies; the static stabilization energy is indeed small and, thus, has not significantly affected the results.

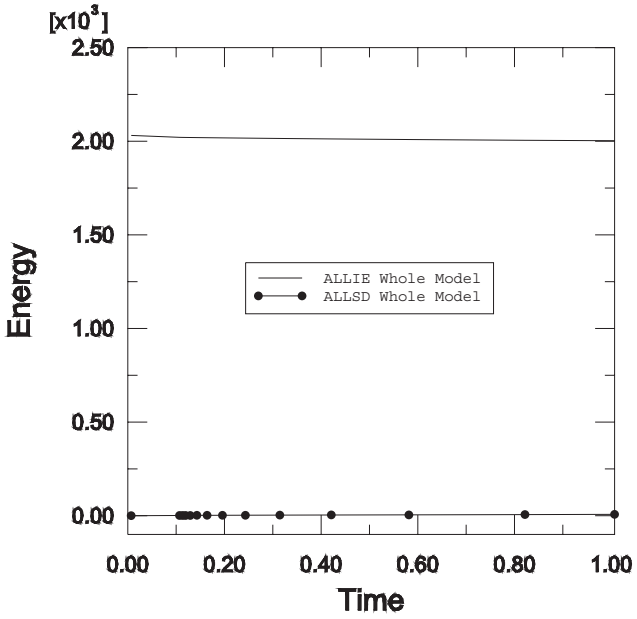


Figure 2: Internal and static stabilization energy histories.

Summary

This section presents a summary of the concepts.

- If a quasi-static analysis is performed in its natural time scale, the solution should be nearly the same as a truly static solution.
- It is often necessary to use load rate scaling or mass scaling to obtain a quasi-static solution using less CPU time.
- The loading rate often can be increased somewhat, as long as the solution does not localize. If the loading rate is increased too much, inertial forces adversely affect the solution.
- Mass scaling is an alternative to increasing the loading rate. When using rate-dependent materials, mass scaling is preferable because increasing the loading rate artificially changes the material properties.
- In a static analysis the lowest modes of the structure dominate the response. Knowing the lowest natural frequency and, correspondingly, the period of the lowest mode, you can estimate the time required to obtain the proper static response.
- It may be necessary to run a series of analyses at varying loading rates to determine an acceptable loading rate.
- The kinetic energy of the deforming material should not exceed a small fraction (typically 5% to 10%) of the internal energy throughout most of the simulation.
- Using a smooth step amplitude curve is the most efficient way to prescribe displacements in a quasi-static analysis.
- Import the model from Abaqus/Explicit to Abaqus/Standard to perform an efficient springback analysis.

Example Files

This appendix contains a list of the Python scripts that can be used to create complete models for the examples contained in this guide.
These files are also included with the Abaqus release; you can extract them from the compressed archive files using the **abaqus fetch** utility.

In this section:

- *Overhead hoist frame*
- *Connecting lug*
- *Skew plate*
- *Cargo crane*
- *Cargo crane – dynamic loading*
- *Nonlinear skew plate*
- *Stress wave propagation in a bar*
- *Connecting lug with plasticity*
- *Blast loading on a stiffened plate*
- *Axisymmetric mount*
- *Vibration of a piping system*
- *Forming a channel*
- *Shearing of a lap joint*
- *Circuit board drop test*

Overhead hoist frame

This Python script can be used to create a model for examples contained in this guide.

- gsi_frame_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_frame_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for frame example
#
#

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

mdb.models.changeKey(fromName='Model-1', toName='standard')

##
##  Sketch profile of frame
##
s = mdb.models['standard'].ConstrainedSketch(name='__profile__', sheetSize=4.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-0.4, -0.2), point2=(0.4, 0.4))
s.delete(objectList=(c[18], c[21], c[19], c[20]))
s.ParallelConstraint(entity1=g.findAt((0.0, 0.4)), entity2=g.findAt((0.0,
    -0.2)))
s.HorizontalDimension(vertex1=v.findAt((-0.4, 0.4)), vertex2=v.findAt((0.4,
    0.4)), textPoint=(0.350665062665939, 0.510207295417786), value=1.0)
s.HorizontalDimension(vertex1=v.findAt((-0.4, -0.2)), vertex2=v.findAt((0.4,
```

```
    -0.2)), textPoint=(0.299293786287308, -0.478947371244431), value=2.0)
s.ObliqueDimension(vertex1=v.findAt((0.4, 0.4)), vertex2=v.findAt((0.4,
    -0.2)), textPoint=(0.589653432369232, 0.0547049306333065), value=1.0)
s.ObliqueDimension(vertex1=v.findAt((-1.6, -0.2)), vertex2=v.findAt((0.2,
    0.4)), textPoint=(-0.524880945682526, 0.353907465934753), value=1.0)
session.viewports['Viewport: 1'].view.fitView()
s.Line(point1=(-0.45, 0.533012701892263), point2=(-0.15, -0.333012701892264))
s.CoincidentConstraint(entity1=v.findAt((-0.15, -0.333013)), entity2=g.findAt(
    (0.05, -0.333013)), addUndoState=False)
s.Line(point1=(-0.15, -0.333012701892264), point2=(0.55, 0.533012701892263))
s.breakCurve(curve1=g.findAt((0.05, -0.333013)), point1=(-0.247599363327026,
    -0.332068353891373), curve2=g.findAt((-0.3, 0.1)), point2=(
    -0.158276319503784, -0.259217292070389))
s.EqualLengthConstraint(entity1=g.findAt((-0.3, 0.1)), entity2=g.findAt((0.2,
    0.1)))
s.HorizontalConstraint(entity= g.findAt((-0.45,   -0.284057), ))
s.HorizontalConstraint(entity=g.findAt((0.55, -0.333013)))
s.move(objectList=(g.findAt((-0.7, 0.1),), g.findAt((0.05, 0.533013), ),
    g.findAt((0.8, 0.1),), g.findAt((-0.2, 0.1), ),
        g.findAt(( 0.3, 0.1), ), g.findAt((0.55,-0.333013), ),
    g.findAt((-0.45,-0.333013), )), vector=(-0.01, 0.0))
p = mdb.models['standard'].Part(name='Frame', dimensionality=TWO_D_PLANAR,
    type=DEFORMABLE_BODY)
p = mdb.models['standard'].parts['Frame']
p.BaseWire(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['standard'].parts['Frame']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']
##
##  Create material 'Steel'
##
mdb.models['standard'].Material('Steel')
mdb.models['standard'].materials['Steel'].Elastic(table=((200.E9, 0.3), ))
##
##  Create truss section
##
area1=pi*0.025**2/4.0
mdb.models['standard'].TrussSection(name='FrameSection', material='Steel',
    area=area1)
##
##  Assign truss section
##
e = p.edges
edges = e
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='FrameSection')

a = mdb.models['standard'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a = mdb.models['standard'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the frame
##
p = mdb.models['standard'].parts['Frame']
a.Instance(name='Frame-1', part=p, dependent=ON)
p1 = a.instances['Frame-1']
```

```
p1.translate(vector=(-0.035794, 0.331227, 0.0))
p1.translate(vector=(-0.004206, 0.001786, 0.0))

##
##  Create a static linear perturbation step
##
mdb.models['standard'].StaticLinearPerturbationStep(name='Apply load',
    previous='Initial', description='250kN central load',
    matrixSolver=SOLVER_DEFAULT)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Apply load')
mdb.models['standard'].fieldOutputRequests['F-Output-1'].setValues(
    variables=PRESELECT, region=MODEL)
##
##  Apply concentrated force to bottom center
##
v = a.instances['Frame-1'].vertices
region=((v.findAt(((0.0, 0.0, 0.0), ), ), ), )
mdb.models['standard'].ConcentratedForce(name='Force',
    createStepName='Apply load',
    region=region, cf2=-250.e3)
##
##  Apply encastre bc to bottom left corner
##
region=(v.findAt(((-1.0, 0.0, 0.0), ), ), None, None, None)
mdb.models['standard'].EncastreBC(name='Fixed', createStepName='Initial',
    region=region)
##
##  Apply roller bc to bottom right corner
##
region=(v.findAt(((1.0, 0.0, 0.0), ), ), None, None, None)
mdb.models['standard'].DisplacementBC(name='Roller',
    createStepName='Initial', region=region, u2=0.0)
##
##  Assign global seed
##
p.seedPart(size=1.0)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=T2D2)
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))
##
##  Generate mesh
##
p.generateMesh()
##
##  Create job
##
mdb.Job(name='Frame', model='standard',
    description='Two-dimensional overhead hoist frame')
mdb.jobs['Frame'].setValues(echoPrint=ON, modelPrint=ON, contactPrint=ON,
    historyPrint=ON)

session.viewports['Viewport: 1'].view.fitView()

##
##  Save model database
##
```

```
mdb.saveAs('Frame')

##
##  Create explicit dynamics model
##
mdb.Model(name='explicit', objectToCopy=mdb.models['standard'])

##
##  add density
##
mdb.models['explicit'].materials['Steel'].Density(table=((7800.0, ), ))

a = mdb.models['explicit'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

##
##  replace static step with explicit dynamics step
##  add new history output requests
##
mdb.models['explicit'].ExplicitDynamicsStep(name='Apply load',
    previous='Initial', maintainAttributes=True, timePeriod=0.01)

v = a.instances['Frame-1'].vertices
verts = v.findAt(((0.0, 0.0, 0.0), ))
a.Set(vertices=verts, name='Center')

regionDef=mdb.models['explicit'].rootAssembly.sets['Center']
mdb.models['explicit'].historyOutputRequests['H-Output-1'].setValues(
    variables=('UT', ), region=regionDef, frequency=1)
##
#  change element library to EXPLICIT
##
elemType1 = mesh.ElemType(elemCode=T2D2, elemLibrary=EXPLICIT)
p = mdb.models['explicit'].parts['Frame']
e = p.edges
edges = e
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))

##
##  create job
##
mdb.Job(name='expFrame', model='explicit', type=ANALYSIS,
    description='Two-dimensional overhead hoist frame-explicit dynamics')

a = mdb.models['standard'].rootAssembly
a.regenerate()
a = mdb.models['explicit'].rootAssembly
a.regenerate()

mdb.save()
```

Connecting lug

This Python script can be used to create a model for examples contained in this guide.

- gsi_lug_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_lug_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for elastic lug example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

mdb.models.changeKey(fromName='Model-1', toName='standard')

session.viewports['Viewport: 1'].setValues(displayedObject=None)

##  Sketch profile of the lug

s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.25)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-0.0275, 0.0225), point2=(0.02125, -0.01375))
s.delete(objectList=(g.findAt((0.02125, 0.004375)), ))
s.EqualLengthConstraint(entity1=g.findAt((-0.003125, -0.01375)),
    entity2=g.findAt((-0.003125, 0.0225)))
s.ObliqueDimension(vertex1=v.findAt((-0.0275, -0.01375)), vertex2=v.findAt((
    0.02125, -0.01375)), textPoint=(-0.010330107063055, -0.023654306307435),
```

```
    value=0.1)
s.move(vector=(-0.025625, 0.0), objectList=(g.findAt((-0.0275, 0.004375)),
    g.findAt((0.0225, -0.01375)), g.findAt((0.0225, 0.0225))))
session.viewports['Viewport: 1'].view.fitView()
s.ObliqueDimension(vertex1=v.findAt((-0.053125, 0.0225)), vertex2=v.findAt((
    -0.053125, -0.01375)), textPoint=(-0.0564765408635139,
    0.00896133482456207), value=0.05)
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(nearPlane=0.230597,
    farPlane=0.279546, width=0.144889, height=0.121934)
s.Arc3Points(point1=(0.046875, 0.03625), point2=(0.046875, -0.01375), point3=(
    0.0625, 0.02375))
s.TangentConstraint(entity1=g.findAt((-0.003125, 0.03625)), entity2=g.findAt((
    0.0657, 0.01125)))
s.move(vector=(-0.001797, 0.0), objectList=(g.findAt((-0.053125, 0.01125)),
   g.findAt((-0.003125, -0.01375)), g.findAt((-0.003125, 0.03625)), g.findAt((

    0.071875, 0.01125))))
s.TangentConstraint(entity1=g.findAt((-0.004922, -0.01375)), entity2=g.findAt((

    0.070078, 0.01125)))
session.viewports['Viewport: 1'].view.fitView()
s.RadialDimension(curve=g.findAt((0.070078, 0.01125)), textPoint=(
    0.0655877739191055, -0.0193991288542748), radius=0.025)
d[2].setValues(reference=ON)
s.CircleByCenterPerimeter(center=(0.0425, 0.0125), point1=(0.0525, 0.01625))
s.ConcentricConstraint(entity1=g.findAt((0.0325, 0.00875)), entity2=g.findAt((
    0.070078, 0.01125)))
s.RadialDimension(curve=g.findAt((0.035078, 0.0075)), textPoint=(
    0.0297759883105755, 0.0219836011528969), radius=0.015)
s.VerticalDimension(vertex1=v.findAt((0.045078, 0.01125)), vertex2=v.findAt((
    0.059123, 0.016517)), textPoint=(0.0632773488759995, 0.0129371425136924),
    value=0.0)
s.VerticalDimension(vertex1=v.findAt((0.045078, 0.01125)), vertex2=v.findAt((
    0.067003, 0.023263)), textPoint=(0.0746369957923889, 0.012744665145874),
    value=0.0)
s.move(vector=(0.034922, 0.00375), objectList=(g.findAt((-0.054922, 0.01125)),

    g.findAt((-0.004922, -0.01375)), g.findAt((-0.004922, 0.03625)),
    g.findAt((0.070078, 0.01125)), g.findAt((0.030078, 0.01125))))
p = mdb.models['standard'].Part(name='Lug', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['standard'].parts['Lug']
p.BaseSolidExtrude(sketch=s, depth=0.02)
s.unsetPrimaryObject()
p = mdb.models['standard'].parts['Lug']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums


##
##  Create material 'Steel'
##
mdb.models['standard'].Material('Steel')
mdb.models['standard'].materials['Steel'].Elastic(table=((200.0E9, 0.3), ))
##
##  Create solid section
##
mdb.models['standard'].HomogeneousSolidSection(name='LugSection',
```

```
    material='Steel', thickness=1.0)
##
##  Assign solid section
##
c = p.cells
cells = c
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='LugSection', offset=0.0)
##
##  Set coordinate system (done by default)
##
a = mdb.models['standard'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the lug
##
a.Instance(name='Lug-1', part=p, dependent=ON)
##
##  Create a static general step
##
mdb.models['standard'].StaticStep(name='LugLoad',
    previous='Initial', description='Apply uniform pressure to the hole')
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='LugLoad')
##
##  Modify output requests
##
mdb.models['standard'].fieldOutputRequests['F-Output-1'].setValues(
    variables=('S', 'U', 'RF', 'NFORC'), region=MODEL)
del mdb.models['standard'].historyOutputRequests['H-Output-1']
##
##  Apply encastre bc to left end
##
f = a.instances['Lug-1'].faces
faces = f.findAt(((-0.02, 0.006667, 0.013333), ))
region = regionToolset.Region(faces=faces)
mdb.models['standard'].EncastreBC(name='Fix left end',
    createStepName='LugLoad', region=region)

session.viewports['Viewport: 1'].setValues(displayedObject=a)
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
##
##  Partition the lug
##
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

pickedCells = c.findAt(((-0.02, 0.006667, 0.013333), ))
p.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(0.095, 0.015,
    0.02)), cells=pickedCells, point2=p.InterestingPoint(edge=e.findAt(
    coordinates=(0.08, 0.03, 0.02)), rule=MIDDLE), point3=p.InterestingPoint(
    edge=e.findAt(coordinates=(0.08, 0.03, 0.0)), rule=MIDDLE))
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Apply pressure load to bottom of hole
##
a.regenerate()
s = a.instances['Lug-1'].faces
side1Faces = s.findAt(((0.094968, 0.01402, 0.013333), ))
region = regionToolset.Region(side1Faces=side1Faces)
mdb.models['standard'].Pressure(name='Pressure load',
```

```
    createStepName='LugLoad', region=region, magnitude=5.0E7)


##
##  Create partitions
##
pickedCells = c.findAt(((-0.02, 0.006667, 0.006667), ), ((-0.02, 0.023333,
    0.013333), ))
p.PartitionCellByPlaneThreePoints(cells=pickedCells, point1=p.InterestingPoint(

    edge=e.findAt(coordinates=(0.069393, 0.004393, 0.0)), rule=MIDDLE),
    point2=p.InterestingPoint(edge=e.findAt(coordinates=(0.069393, 0.004393,
    0.02)), rule=MIDDLE), point3=p.InterestingPoint(edge=e.findAt(
    coordinates=(0.090607, 0.025607, 0.02)), rule=MIDDLE))

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

p.DatumPointByEdgeParam(edge=e.findAt(coordinates=(0.055, -0.01, 0.02)),
    parameter=0.25)
pickedCells = c.findAt(((-0.02, 0.006667, 0.006667), ), ((-0.02, 0.023333,
    0.013333), ))
p.PartitionCellByPlanePointNormal(point=d[5], normal=e.findAt(coordinates=(
    0.055, -0.01, 0.02)), cells=pickedCells)
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Assign global seed
##
p.seedPart(size=0.007)
import re, uti
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=0.01)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=C3D20R)
elemType2 = mesh.ElemType(elemCode=C3D15)
elemType3 = mesh.ElemType(elemCode=C3D10M)
cells = c
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))
##
##  Generate the mesh
##
p.generateMesh()
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
    datumPoints=OFF)
##
##  Create job
##
a.regenerate()
mdb.Job(name='Lug', model='standard',
    description='Linear Elastic Steel Connecting Lug')
##
##  Save model database
##
mdb.saveAs('Lug.cae')
##
```

```
##  Explicit dynamics model
##
mdb.Model(name='explicit', objectToCopy=mdb.models['standard'])

##
##  add density
##
mdb.models['explicit'].materials['Steel'].Density(table=((7800.0, ), ))

##
##  replace static step with explicit dynamics step
##
a = mdb.models['explicit'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
mdb.models['explicit'].ExplicitDynamicsStep(name='LugLoad',
    previous='Initial', maintainAttributes=True, timePeriod=0.005)

##
##  change number of field output intervals to 125
##
mdb.models['explicit'].fieldOutputRequests['F-Output-1'].setValues(
    numIntervals=125)

##
##  change element type to linear and library to EXPLICIT
##
elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
    hourglassControl=ENHANCED)
elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
p = mdb.models['explicit'].parts['Lug']
c = p.cells
cells = c
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))

pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=0.007)
   p.generateMesh()

##
##  create job
##
mdb.Job(name='expLug', model='explicit', type=ANALYSIS,
    description='dynamic analysis of lug')

a = mdb.models['standard'].rootAssembly
a.regenerate()
a = mdb.models['explicit'].rootAssembly
a.regenerate()

mdb.save()
```

Skew plate

This Python script can be used to create a model for examples contained in this guide.

- gsi_skewplate_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_skewplate_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for linear skew plate example
#

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

##
##  Sketch profile of the plate
##
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=4.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-0.275, 0.25), point2=(0.575, -0.25))
s.delete(objectList=(c[18], c[19], c[20], c[21], c[22]))
s.VerticalConstraint(entity=g.findAt((-0.275, 0.0)))
s.VerticalConstraint(entity=g.findAt((0.575, 0.0)))
s.ParallelConstraint(entity1=g.findAt((0.15, -0.25)), entity2=g.findAt((0.15,
    0.25)))
s.FixedConstraint(entity=g.findAt((0.15, -0.25)))
s.ObliqueDimension(vertex1=v.findAt((-0.275, 0.25)), vertex2=v.findAt((-0.275,

    -0.25)), textPoint=(-0.384168207645416, -0.039074968546629), value=0.4)
```

```
s.delete(objectList=(c[26], ))
s.HorizontalDimension(vertex1=v.findAt((-0.275, -0.25)), vertex2=v.findAt((
    0.575, -0.25)), textPoint=(0.502545475959778, -0.405263155698776),
    value=1.0)
s.AngularDimension(line1=g.findAt((-0.275, -0.05)), line2=g.findAt((0.225,
    -0.25)), textPoint=(-0.221120104193687, -0.173046261072159), value=60.0)
session.viewports['Viewport: 1'].view.fitView()
s.move(vector=(-0.725, 0.05), objectList=(g.findAt((-0.275, -0.05)), g.findAt((

    0.225, 0.038675)), g.findAt((0.725, 0.52735)), g.findAt((0.225,
    0.438675))))
p = mdb.models['Model-1'].Part(name='Plate', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Plate']
p.BaseShell(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Plate']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
##
##  Create material 'Metal'
##
mdb.models['Model-1'].Material('Metal')
mdb.models['Model-1'].materials['Metal'].Elastic(table=((30.0E9, 0.3), ))
##
##  Create shell section
##
mdb.models['Model-1'].HomogeneousShellSection(name='PlateSection',
    preIntegrate=ON,
    material='Metal', thickness=0.008,
    poissonDefinition=DEFAULT, temperature=GRADIENT)
##
##  Create datum coordinate system
##
p1 = mdb.models['Model-1'].parts['Plate']
e = p1.edges
p1.DatumCsysByTwoLines(CARTESIAN, line1=e.findAt((-0.75,
    -0.0556624327025937, 0.0), ), line2=e.findAt((0.0,
    0.477350269189625, 0.0), ))
##
##  Assign material orientation to the plate
##
p0 = mdb.models['Model-1'].parts['Plate']
f = p0.faces
faces = f[0:1]
region=(None, None,
    f.findAt(((-0.666666666666667,
    0.125783423063208, 0.0), (0.0, 0.0, 1.0)), ), None)
datum = p0.datums[2]
p0.MaterialOrientation(region=region, localCsys=datum, axis=AXIS_3)
##
##  Assign section to the plate
##
p0 = mdb.models['Model-1'].parts['Plate']
f = p0.faces
faces = f[0:1]
region=(None, None,
    f.findAt(((-0.666666666666667,
    0.125783423063208, 0.0), (0.0, 0.0, 1.0)), ), None)
p0 = mdb.models['Model-1'].parts['Plate']
p0.SectionAssignment(region=region, sectionName='PlateSection')
```

```
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the plate
##
p = mdb.models['Model-1'].parts['Plate']
a.Instance(name='Plate-1', part=p, dependent=ON)
##
##  Partition the plate
##
p = mdb.models['Model-1'].parts['Plate']
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

faces=(f.findAt((
    -0.666666666666667, 0.125783423063208, 0.0), (0.0, 0.0, 1.0)), )
p.PartitionFaceByShortestPath(faces=faces,
    point1=p.InterestingPoint(e.findAt((-0.75,
    -0.0556624327025937, 0.0), ), MIDDLE),
    point2=p.InterestingPoint(e.findAt((-0.25,
    0.63301270189222, 0.0), ), MIDDLE))

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Create geometry set 'MidSpan'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (-0.5, 0.188675134594813, 0.0), ), ), name='MidSpan')
##
##  Create geometry set 'EndA'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (-1.0, 0.1, 0.0), ), ), name='EndA')
##
##  Create geometry set 'EndB'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (0.0, 0.477350269189625, 0.0), ), ), name='EndB')

##
##  Create a static general step
##
mdb.models['Model-1'].StaticStep(name='Apply pressure',
    previous='Initial', description='Uniform pressure(20 kPa) load',
    timePeriod=1, adiabatic=OFF, maxNumInc=100,
    stabilization=None, timeIncrementationMethod=AUTOMATIC, initialInc=1,
    minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT, amplitude=RAMP,
    extrapolation=LINEAR, fullyPlastic="")
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
```

```
    step='Apply pressure')
##
##  Modify output requests
##
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=('S', 'E', 'U', 'RF'))
a0=mdb.models['Model-1'].rootAssembly
regionDef=a0.sets['MidSpan']
mdb.models['Model-1'].historyOutputRequests['H-Output-1'].setValues(
    variables=('U3',), region=regionDef)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
##
##  Apply bc in local CSYS to set EndB (right edge)
##
a0 = mdb.models['Model-1'].rootAssembly
region = a0.sets['EndB']
datum = mdb.models['Model-1'].rootAssembly.instances['Plate-1'].datums[2]
mdb.models['Model-1'].DisplacementBC(name='Rail boundary condition',
    createStepName='Apply pressure', region=region, u2=0.0, u3=0.0, ur1=0.0,
    ur2=0.0, ur3=0.0, localCsys=datum)
##
##  Apply encastre bc to set EndA (left edge)
##
a0 = mdb.models['Model-1'].rootAssembly
region = a0.sets['EndA']
mdb.models['Model-1'].EncastreBC(name='Fix left end',
    createStepName='Apply pressure', region=region)
##
##  Apply pressure load
##
a0 = mdb.models['Model-1'].rootAssembly
f1 = a0.instances['Plate-1'].faces
region=((
    f1.findAt(((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)), ((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)), ), SIDE1),
    )
mdb.models['Model-1'].Pressure(name='Pressure',
    createStepName='Apply pressure', region=region, magnitude=2.0E4)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON, loads=OFF,
    bcs=OFF, predefinedFields=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
##
##  Assign global seed
##
p.seedPart(size=0.1)
##
##  Use structured meshing
##
f1 = p.faces
regions=(
    f1.findAt((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)),
    f1.findAt((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)))
p.setMeshControls(regions=regions, technique=STRUCTURED)
##
```

```
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=S8R5)
elemType2 = mesh.ElemType(elemCode=STRI65)
regions=(None, None,
    f1.findAt(((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)), ((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)), ), None)
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2))
##
##  Generate mesh
##
p.generateMesh()

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
##
##  Create job
##
mdb.Job(name='SkewPlate', model='Model-1',
    description='Linear Elastic Skew Plate. 20 kPa Load.')

a = mdb.models['Model-1'].rootAssembly
a.regenerate()

##
##  Save model database
##
mdb.saveAs('SkewPlate.cae')
```

Cargo crane

This Python script can be used to create a model for examples contained in this guide.

- gsi_crane_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_crane_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for static crane example
#


def GetBlockPosition(modelName, blockPrefix):
    if blockPrefix == '':
        return len(mdb.models[modelName].keywordBlock.sieBlocks)-1
    pos = 0
    for block in mdb.models[modelName].keywordBlock.sieBlocks:
        if block[0:len(blockPrefix)].lower()==blockPrefix.lower():
            return pos
        pos=pos+1
    return -1

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
Mdb()

session.viewports['Viewport: 1'].setValues(displayedObject=None)
##
##  Sketch profile of truss
##
p = mdb.models['Model-1'].Part(name='Truss', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p.ReferencePoint(point=(0.0, 0.0, 0.0))
```

```
p = mdb.models['Model-1'].parts['Truss']
session.viewports['Viewport: 1'].setValues(displayedObject=p)

d, r = p.datums, p.referencePoints
p.DatumPointByOffset(point=r[1], vector=(0.0, 1.0, 0.0))
p.DatumPointByOffset(point=r[1], vector=(8.0, 1.5, 0.9))
p.DatumPlaneByThreePoints(point1=r[1], point2=d[3], point3=d[2])
p.DatumAxisByPrincipalAxis(principalAxis=YAXIS)

session.viewports['Viewport: 1'].view.fitView()

t = p.MakeSketchTransform(sketchPlane=d[4], sketchUpEdge=d[5],
    sketchPlaneSide=SIDE1, sketchOrientation=LEFT, origin=(0.0, 0.0, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=683.38, gridSpacing=17.08, transform=t)
g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Truss']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
session.viewports['Viewport: 1'].view.setValues(nearPlane=648.075,
    farPlane=719.271, width=13.6266, height=9.28111, cameraPosition=(-71.5628,

    0.978423, 679.645), cameraTarget=(4.83613, 0.978423, 0.544065))

s.Line(point1=(0.0, 0.0), point2=(8.05046582503149, 1.5))
s.CoincidentConstraint(entity1=v.findAt((0.0, 0.0)), entity2=g.findAt((0.0,
    0.5)))

s.Line(point1=(8.05046582503149, 1.5), point2=(0.0, 1.0))

s.Line(point1=(0.0, 1.0), point2=(1.59409310185916, 0.297018794285658))
s.CoincidentConstraint(entity1=v.findAt((1.594093, 0.297019)),
    entity2=g.findAt((4.025233, 0.75)))

s.Line(point1=(1.59409310185916, 0.297018794285658),
    point2=(2.30592731982451, 1.14321701190598))
s.CoincidentConstraint(entity1=v.findAt((2.305927, 1.143217)),
    entity2=g.findAt((4.025233, 1.25)))

s.Line(point1=(2.30592731982451, 1.14321701190598),
    point2=(3.26033977276254, 0.607481575033539))
s.CoincidentConstraint(entity1=v.findAt((3.26034, 0.607482)),
    entity2=g.findAt((4.025233, 0.75)))

s.Line(point1=(3.26033977276254, 0.607481575033539),
    point2=(4.16525059699136, 1.25869624736747))
s.CoincidentConstraint(entity1=v.findAt((4.165251, 1.258696)),
    entity2=g.findAt((4.025233, 1.25)))

s.Line(point1=(4.16525059699136, 1.25869624736747),
    point2=(4.95150583156061, 0.922587451305882))
s.CoincidentConstraint(entity1=v.findAt((4.951506, 0.922587)),
    entity2=g.findAt((4.025233, 0.75)))

s.breakCurve(curve1=g.findAt((4.025233, 0.75)),
    point1=(0.733189206279801, 0.182644009590149),
    curve2=g.findAt((1.95001, 0.720118)),
    point2=(1.62787567272877, 0.433001518249512))

s.breakCurve(curve1=g.findAt((4.822279, 0.898509)),
```

```
    point1=(2.34362044857835, 0.415118813514709),
    curve2=g.findAt((2.783134, 0.875349)),
    point2=(3.22040910681461, 0.701241612434387))

s.breakCurve(curve1=g.findAt((5.655403, 1.053741)),
    point1=(4.04351950354929, 0.772772312164307),
    curve2=g.findAt((4.558378, 1.090642)),
    point2=(5.02767302451382, 0.95159912109375))

s.breakCurve(curve1=g.findAt((4.025233, 1.25)),
    point1=(1.55629884485765, 1.09466052055359),
    curve2=g.findAt((1.95001, 0.720118)),
    point2=(2.32572264036564, 1.02312982082367))

s.breakCurve(curve1=g.findAt((5.178197, 1.321609)),
    point1=(3.79300780849022, 1.23772192001343),
    curve2=g.findAt((3.712795, 0.933089)),
    point2=(4.168774592922, 1.18407392501831))

s.VerticalDimension(vertex1=v.findAt((0.0, 1.0)),
    vertex2=v.findAt((0.0, 0.0)),
    textPoint=(-0.304643334343144, 0.0216999053955078), value=1.0)
s.HorizontalDimension(vertex1=v.findAt((0.0, 0.0)),
    vertex2=v.findAt((8.050466, 1.5)),
    textPoint=(6.26233672422377, -0.443249702453613), value=8.05046582503149)

s.ParallelConstraint(entity1=g.findAt((1.152964, 1.071609)), entity2=g.findAt((

    3.235589, 1.200957)))

s.ParallelConstraint(entity1=g.findAt((3.235589, 1.200957)), entity2=g.findAt((

    6.107858, 1.379348)))

s.ParallelConstraint(entity1=g.findAt((0.797047, 0.148509)), entity2=g.findAt((

    2.427216, 0.45225)))

s.ParallelConstraint(entity1=g.findAt((2.427216, 0.45225)), entity2=g.findAt((
    4.105923, 0.765035)))

s.ParallelConstraint(entity1=g.findAt((4.105923, 0.765035)), entity2=g.findAt((

    6.500986, 1.211294)))

s.EqualLengthConstraint(entity1=g.findAt((0.797047, 0.148509)),
    entity2=g.findAt((2.427216, 0.45225)))

s.EqualLengthConstraint(entity1=g.findAt((2.39114, 0.445528)),
    entity2=g.findAt((4.033769, 0.751591)))

s.EqualLengthConstraint(entity1=g.findAt((3.985233, 0.742547)),
    entity2=g.findAt((6.416373, 1.195528)))

s.EqualLengthConstraint(entity1=g.findAt((1.152964, 1.071609)),
    entity2=g.findAt((3.235589, 1.200957)))

s.EqualLengthConstraint(entity1=g.findAt((3.458891, 1.214826)),
    entity2=g.findAt((6.33116, 1.393217)))

mdb.models['Model-1'].ConstrainedSketch(name='Truss', objectToCopy=s)
```

```
p = mdb.models['Model-1'].parts['Truss']
d = p.datums
p.Wire(sketchPlane=d[4], sketchUpEdge=d[5], sketchPlaneSide=SIDE1,
    sketchOrientation=LEFT, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']

session.viewports['Viewport: 1'].view.fitView()

p = mdb.models['Model-1'].parts['Truss']
d, r = p.datums, p.referencePoints
p.DatumPointByOffset(point=d[2], vector=(0.0, 0.0, 2.0))
p.DatumPointByOffset(point=r[1], vector=(0.0, 0.0, 2.0))
p.DatumPointByOffset(point=d[3], vector=(0.0, 0.0, 0.2))
p.DatumPlaneByThreePoints(point1=d[8], point2=d[9], point3=d[7])

t = p.MakeSketchTransform(sketchPlane=d[10], sketchUpEdge=d[5],
    sketchPlaneSide=SIDE1, sketchOrientation=LEFT, origin=(4.122203, 0.75,
    1.536252))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=683.38, gridSpacing=17.08, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.retrieveSketch(sketch=mdb.models['Model-1'].sketches['Truss'])
s.move(vector=(-4.14820681508444, -0.75), objectList=(g.findAt((0.0, 0.5)),
    g.findAt((1.006308, 0.6875)), g.findAt((2.348053, 0.770833)), g.findAt((
    3.354361, 0.958333)), g.findAt((4.696105, 1.041667)), g.findAt((5.702413,
    1.229167)), g.findAt((1.006308, 0.1875)), g.findAt((3.018925, 0.5625)),
    g.findAt((5.031541, 0.9375)), g.findAt((7.044158, 1.3125)), g.findAt((
    1.341744, 1.083333)), g.findAt((6.708722, 1.416667)), g.findAt((4.025233,
    1.25)), v.findAt((8.050466, 1.5)), v.findAt((0.0, 1.0))))
d = p.datums
p.Wire(sketchPlane=d[10], sketchUpEdge=d[5], sketchPlaneSide=SIDE1,
    sketchOrientation=LEFT, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']

##
##  dummy part
##
p = mdb.models['Model-1'].Part(name='Truss-all',
    objectToCopy=mdb.models['Model-1'].parts['Truss'])
session.viewports['Viewport: 1'].setValues(displayedObject=p)
v = p.vertices
p.WirePolyLine(points=((v.findAt(coordinates=(2.0, 0.375, 1.775)), v.findAt(
    coordinates=(2.0, 0.375, 0.225))), (v.findAt(coordinates=(2.0, 0.375,
    0.225)), v.findAt(coordinates=(2.666667, 1.166667, 1.7))), (v.findAt(
    coordinates=(2.666667, 1.166667, 1.7)), v.findAt(coordinates=(2.666667,
    1.166667, 0.3))), (v.findAt(coordinates=(4.0, 0.75, 1.55)), v.findAt(
    coordinates=(4.0, 0.75, 0.45))), (v.findAt(coordinates=(4.0, 0.75, 0.45)),

    v.findAt(coordinates=(5.333333, 1.333333, 1.4))), (v.findAt(coordinates=(
    5.333333, 1.333333, 1.4)), v.findAt(coordinates=(5.333333, 1.333333,
    0.6))), (v.findAt(coordinates=(5.333333, 1.333333, 0.6)), v.findAt(
    coordinates=(6.0, 1.125, 1.325))), (v.findAt(coordinates=(6.0, 1.125,
    1.325)), v.findAt(coordinates=(6.0, 1.125, 0.675)))), mergeType=IMPRINT,
    meshable=ON)
##
##  create cross brace part in the assembly module
```

```
##

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##  Set coordinate system (done by default)
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Truss']
a.Instance(name='Truss-1', part=p, dependent=ON)
p = mdb.models['Model-1'].parts['Truss-all']
a.Instance(name='Truss-all-1', part=p, dependent=ON)
a.PartFromBooleanCut(name='Cross brace',
    instanceToBeCut=a.instances['Truss-all-1'],
    cuttingInstances=(a.instances['Truss-1'], ))
p = mdb.models['Model-1'].parts['Cross brace']
a.Instance(name='Cross brace-1', part=p, dependent=ON)
a.suppressFeatures(('Truss-1', 'Truss-all-1', ))
a.features['Truss-1'].resume()

p = mdb.models['Model-1'].parts['Cross brace']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
##
##  Create beam profiles and beam sections
##
mdb.models['Model-1'].BoxProfile(name='MainBoxProfile', b=0.05,
    a=0.1, uniformThickness=ON, t1=0.005)
mdb.models['Model-1'].BoxProfile(name='BraceBoxProfile', b=0.03,
    a=0.03, uniformThickness=ON, t1=0.003)
mdb.models['Model-1'].BeamSection(name='MainMemberSection',
    profile='MainBoxProfile', poissonRatio=0.25,
    integration=BEFORE_ANALYSIS, table=((2.e11, 8.e10),))
mdb.models['Model-1'].BeamSection(name='BracingSection',
    profile='BraceBoxProfile', poissonRatio=0.25,
    integration=BEFORE_ANALYSIS, table=((2.e11, 8.e10),))
##
##  Flip tangent directions
##
##p = mdb.models['Model-1'].parts['Truss']
##e = p.edges
##edges = e
##regions = regionToolset.Region(edges=edges)
##p.flipTangent(regions=regions)
####
##edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((2.5, 0.46875, 0.28125), ),
((
##    0.5, 0.09375, 0.05625), ), ((4.5, 0.84375, 1.49375), ), ((2.5, 0.46875,
##    1.71875), ), ((0.5, 0.09375, 1.94375), ))
##regions = regionToolset.Region(edges=edges)
##p.flipTangent(regions=regions)
##
##  Assign beam sections to the cross brace
##
p = mdb.models['Model-1'].parts['Cross brace']
e = p.edges
edges = e.findAt(((2.666667, 1.166667, 1.35), ),
    ((2.166667, 0.572917, 0.59375), ), ((2.0, 0.375, 1.3875), ),
    ((6.0, 1.125, 1.1625), ), ((5.5, 1.28125, 0.78125), ),
    ((5.333333, 1.333333, 1.2), ), ((4.333333, 0.895833, 0.6875), ),
    ((4.0, 0.75, 1.275), ))
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='BracingSection')
##
```

```
##  Assign beam sections to the internal bracing
##
p = mdb.models['Model-1'].parts['Truss']
e = p.edges
edges = e.findAt(((3.0, 1.0625, 0.3375), ), ((5.5, 1.28125, 0.61875), ),
    ((4.333333, 0.895833, 0.4875), ), ((0.5, 0.84375, 0.05625), ),
    ((2.166667, 0.572917, 0.24375), ), ((3.0, 1.0625, 1.6625), ),
    ((5.5, 1.28125, 1.38125), ), ((4.333333, 0.895833, 1.5125), ),
    ((0.5, 0.84375, 1.94375), ), ((2.166667, 0.572917, 1.75625), ))
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='BracingSection')
##
##  Assign beam sections to the main members
##
p = mdb.models['Model-1'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((4.666667, 1.291667, 0.525), ),

    ((6.5, 1.21875, 0.73125), ), ((7.333333, 1.458333, 0.825), ),
    ((2.5, 0.46875, 0.28125), ), ((2.0, 1.125, 0.225), ),
    ((0.5, 0.09375, 0.05625), ), ((4.5, 0.84375, 1.49375), ),
    ((4.666667, 1.291667, 1.475), ), ((6.5, 1.21875, 1.26875), ),
    ((7.333333, 1.458333, 1.175), ), ((2.5, 0.46875, 1.71875), ),
    ((2.0, 1.125, 1.775), ), ((0.5, 0.09375, 1.94375), ))
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='MainMemberSection')
##
##  Assign beam section orientations to truss 'B'
##
p = mdb.models['Model-1'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((3.0, 1.0625, 0.3375), ),
    ((4.666667, 1.291667, 0.525), ), ((5.5, 1.28125, 0.61875), ),
    ((6.5, 1.21875, 0.73125), ), ((7.333333, 1.458333, 0.825), ),
    ((4.333333, 0.895833, 0.4875), ), ((2.5, 0.46875, 0.28125), ),
    ((0.5, 0.84375, 0.05625), ), ((2.0, 1.125, 0.225), ),
    ((2.166667, 0.572917, 0.24375), ), ((0.5, 0.09375, 0.05625), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(-0.1118,
    0.0, 0.9936))
##
##  Assign beam section orientations to truss 'A'
##
p = mdb.models['Model-1'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 1.49375), ), ((3.0, 1.0625, 1.6625), ),
    ((4.666667, 1.291667, 1.475), ), ((5.5, 1.28125, 1.38125), ),
    ((6.5, 1.21875, 1.26875), ), ((7.333333, 1.458333, 1.175), ),
    ((4.333333, 0.895833, 1.5125), ), ((2.5, 0.46875, 1.71875), ),
    ((0.5, 0.84375, 1.94375), ), ((2.0, 1.125, 1.775), ),
    ((2.166667, 0.572917, 1.75625), ), ((0.5, 0.09375, 1.94375), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(-0.1118,
    0.0, -0.9936))
##
##  Assign beam section orientations to 'Cross brace'
##
p = mdb.models['Model-1'].parts['Cross brace']
e = p.edges
edges = e.findAt(((2.666667, 1.166667, 1.35), ), ((2.166667, 0.572917,
    0.59375), ), ((2.0, 0.375, 1.3875), ), ((6.0, 1.125, 1.1625), ), ((5.5,
```

```
    1.28125, 0.78125), ), ((5.333333, 1.333333, 1.2), ), ((4.333333, 0.895833,
    0.6875), ), ((4.0, 0.75, 1.275), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(0.0, 1.0, 0.0))

session.viewports['Viewport: 1'].setValues(displayedObject=a)
a.regenerate()
##
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', 'Cross brace-1'))
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
   geometryEdgesInShaded=OFF, datumPoints=OFF, datumAxes=OFF, datumPlanes=OFF,
   datumCoordSystems=OFF, referencePointLabels=OFF, referencePointSymbols=OFF)
##
##  Translate the truss
##
p = a.instances['Truss-1']
p.translate(vector=(0.0, -0.5, -1.0))
##
##  Translate the cross brace
##
p = a.instances['Cross brace-1']
p.translate(vector=(0.0, -0.5, -1.0))


##
##  Create geometry sets
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', ))
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
    datumPoints=OFF)

v = a.instances['Truss-1'].vertices
verts = v.findAt(((0.0, 0.5, -1.0), ), ((0.0, -0.5, -1.0), ), ((0.0, 0.5,
    1.0), ), ((0.0, -0.5, 1.0), ))
a.Set(vertices=verts, name='Attach')

verts = v.findAt(((8.0, 1.0, 0.1), ))
a.Set(vertices=verts, name='Tip-a')

verts = v.findAt(((8.0, 1.0, -0.1), ))
a.Set(vertices=verts, name='Tip-b')

e = a.instances['Truss-1'].edges
edges = e.findAt(((6.5, 0.71875, 0.26875), ))
a.Set(edges=edges, name='Leg-a')

edges = e.findAt(((6.5, 0.71875, -0.26875), ))
a.Set(edges=edges, name='Leg-b')

edges = e.findAt(((4.333333, 0.395833, 0.5125), ))
a.Set(edges=edges, name='Inner-a')

edges = e.findAt(((4.333333, 0.395833, -0.5125), ))
a.Set(edges=edges, name='Inner-b')
```

```
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Cross brace-1', ))
session.viewports['Viewport: 1'].view.setValues(
    cameraPosition=(7.4736, 8.3559, 20.002),
    cameraUpVector=(-0.34023, 0.72044, -0.60432))

session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', 'Cross brace-1'))
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])

##
##  Create a static general step
##
mdb.models['Model-1'].StaticStep(name='Tip load',
    previous='Initial', description='Static tip load on crane',
    timePeriod=1, adiabatic=OFF, maxNumInc=100,
    stabilization=None, timeIncrementationMethod=AUTOMATIC, initialInc=1,
    minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT, amplitude=RAMP,
    extrapolation=LINEAR, fullyPlastic="")
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Tip load')
##
##  Modify output requests
##
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=('U', 'RF', 'SF'))
mdb.models['Model-1'].historyOutputRequests['H-Output-1'].setValues(
    variables=PRESELECT)

##
##  Create constraints between the tips of the truss
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(interactions=ON)
mdb.models['Model-1'].Equation(name='TipConstraint-1',
    terms=((1.0, 'Tip-a', 1), (-1.0, 'Tip-b', 1)))
mdb.models['Model-1'].Constraint('TipConstraint-2',
    mdb.models['Model-1'].constraints['TipConstraint-1'])
mdb.models['Model-1'].constraints['TipConstraint-2'].setValues(
    terms=((1.0, 'Tip-a', 2), (-1.0, 'Tip-b', 2)))
##
##  Create JOIN connectors between truss and bracing
##

v1 = a.instances['Cross brace-1'].vertices
v2 = a.instances['Truss-1'].vertices

a.WirePolyLine(points=((v1.findAt(coordinates=(2.0, -0.125, 0.775)), v2.findAt(

    coordinates=(2.0, -0.125, 0.775))), (v1.findAt(coordinates=(2.666667,
    0.666667, 0.7)), v2.findAt(coordinates=(2.666667, 0.666667, 0.7))), (
    v1.findAt(coordinates=(2.0, -0.125, -0.775)), v2.findAt(coordinates=(2.0,
    -0.125, -0.775))), (v1.findAt(coordinates=(2.666667, 0.666667, -0.7)),
    v2.findAt(coordinates=(2.666667, 0.666667, -0.7))), (v1.findAt(
    coordinates=(4.0, 0.25, -0.55)), v2.findAt(coordinates=(4.0, 0.25,
    -0.55))), (v1.findAt(coordinates=(4.0, 0.25, 0.55)), v2.findAt(
    coordinates=(4.0, 0.25, 0.55))), (v1.findAt(coordinates=(5.333333,
    0.833333, 0.4)), v2.findAt(coordinates=(5.333333, 0.833333, 0.4))), (
    v1.findAt(coordinates=(5.333333, 0.833333, -0.4)), v2.findAt(coordinates=(
    5.333333, 0.833333, -0.4))), (v1.findAt(coordinates=(6.0, 0.625, -0.325)),

    v2.findAt(coordinates=(6.0, 0.625, -0.325))), (v1.findAt(coordinates=(6.0,
```

```
    0.625, 0.325)), v2.findAt(coordinates=(6.0, 0.625, 0.325)))),
    mergeType=IMPRINT, meshable=OFF)

e1 = a.edges
edges1 = e1.findAt(((6.000025, 0.625, 0.325), ), ((6.000025, 0.625, -0.325),
),
    ((5.333358, 0.833333, -0.4), ), ((5.333358, 0.833333, 0.4), ), ((4.000025,

    0.25, 0.55), ), ((4.000025, 0.25, -0.55), ), ((2.666692, 0.666667, -0.7),
    ), ((2.000025, -0.125, -0.775), ), ((2.666692, 0.666667, 0.7), ), ((
    2.000025, -0.125, 0.775), ))
a.Set(edges=edges1, name='Wire-1-Edge-1')

mdb.models['Model-1'].ConnectorSection(name='ConnSect-1',
    translationalType=JOIN)

region=a.sets['Wire-1-Edge-1']
a.SectionAssignment(sectionName='ConnSect-1', region=region)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON, interactions=OFF)
##
##  Apply encastre bc to set 'Attach'
##
region = a.sets['Attach']
mdb.models['Model-1'].EncastreBC(name='Fixed end', createStepName='Tip load',
    region=region)
##
##  Apply concentrated force to set 'Tip-b'
##
region = a.sets['Tip-b']
mdb.models['Model-1'].ConcentratedForce(name='Tip load',
    createStepName='Tip load', region=region, cf2=-10000.0)
##
##  Assign global seed
##
p = mdb.models['Model-1'].parts['Truss']
p.seedPart(size=2.0)
p = mdb.models['Model-1'].parts['Cross brace']
p.seedPart(size=2.0)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=B33)

p = mdb.models['Model-1'].parts['Cross brace']
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))

p = mdb.models['Model-1'].parts['Truss']
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))
##
##  Generate mesh
##
p = mdb.models['Model-1'].parts['Truss']
p.generateMesh()
```

```
p = mdb.models['Model-1'].parts['Cross brace']
p.generateMesh()

##
##  Add keywords
##
mdb.models['Model-1'].keywordBlock.synchVersions()
mdb.models['Model-1'].keywordBlock.insert(GetBlockPosition('Model-1', '*End
Assembly')-1, """*NORMAL, TYPE=ELEMENT
Inner-a,  Inner-a, -0.3962,  0.9171,  0.0446
Inner-b,  Inner-b,  0.3962, -0.9171,  0.0446
Leg-a,    Leg-a,   -0.1820,  0.9829,  0.0205
Leg-b,    Leg-b,    0.1820, -0.9829,  0.0205""")
##
##  Create job
##
mdb.Job(name='Crane', model='Model-1',
    description='3-D model of light-service cargo crane')

a.regenerate()
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Save model database
##
mdb.saveAs('Crane.cae')
```

Cargo crane – dynamic loading

This Python script can be used to create a model for examples contained in this guide.

- gsi_dyncrane_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_dyncrane_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for dynamic crane example
#


def GetBlockPosition(modelName, blockPrefix):
    if blockPrefix == '':
        return len(mdb.models[modelName].keywordBlock.sieBlocks)-1
    pos = 0
    for block in mdb.models[modelName].keywordBlock.sieBlocks:
        if block[0:len(blockPrefix)].lower()==blockPrefix.lower():
            return pos
        pos=pos+1
    return -1

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
Mdb()

mdb.models.changeKey(fromName='Model-1', toName='standard')
##
##  Sketch profile of truss
##
p = mdb.models['standard'].Part(name='Truss', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p.ReferencePoint(point=(0.0, 0.0, 0.0))
```

```
p = mdb.models['standard'].parts['Truss']
session.viewports['Viewport: 1'].setValues(displayedObject=p)

d, r = p.datums, p.referencePoints
p.DatumPointByOffset(point=r[1], vector=(0.0, 1.0, 0.0))
p.DatumPointByOffset(point=r[1], vector=(8.0, 1.5, 0.9))
p.DatumPlaneByThreePoints(point1=r[1], point2=d[3], point3=d[2])
p.DatumAxisByPrincipalAxis(principalAxis=YAXIS)

session.viewports['Viewport: 1'].view.fitView()

t = p.MakeSketchTransform(sketchPlane=d[4], sketchUpEdge=d[5],
    sketchPlaneSide=SIDE1, sketchOrientation=LEFT, origin=(0.0, 0.0, 0.0))
s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=683.38, gridSpacing=17.08, transform=t)
g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['standard'].parts['Truss']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
session.viewports['Viewport: 1'].view.setValues(nearPlane=648.075,
    farPlane=719.271, width=13.6266, height=9.28111, cameraPosition=(-71.5628,

    0.978423, 679.645), cameraTarget=(4.83613, 0.978423, 0.544065))

s.Line(point1=(0.0, 0.0), point2=(8.05046582503149, 1.5))
s.CoincidentConstraint(entity1=v.findAt((0.0, 0.0)), entity2=g.findAt((0.0,
    0.5)))

s.Line(point1=(8.05046582503149, 1.5), point2=(0.0, 1.0))

s.Line(point1=(0.0, 1.0), point2=(1.59409310185916, 0.297018794285658))
s.CoincidentConstraint(entity1=v.findAt((1.594093, 0.297019)),
    entity2=g.findAt((4.025233, 0.75)))

s.Line(point1=(1.59409310185916, 0.297018794285658),
    point2=(2.30592731982451, 1.14321701190598))
s.CoincidentConstraint(entity1=v.findAt((2.305927, 1.143217)),
    entity2=g.findAt((4.025233, 1.25)))

s.Line(point1=(2.30592731982451, 1.14321701190598),
    point2=(3.26033977276254, 0.607481575033539))
s.CoincidentConstraint(entity1=v.findAt((3.26034, 0.607482)),
    entity2=g.findAt((4.025233, 0.75)))

s.Line(point1=(3.26033977276254, 0.607481575033539),
    point2=(4.16525059699136, 1.25869624736747))
s.CoincidentConstraint(entity1=v.findAt((4.165251, 1.258696)),
    entity2=g.findAt((4.025233, 1.25)))

s.Line(point1=(4.16525059699136, 1.25869624736747),
    point2=(4.95150583156061, 0.922587451305882))
s.CoincidentConstraint(entity1=v.findAt((4.951506, 0.922587)),
    entity2=g.findAt((4.025233, 0.75)))

s.breakCurve(curve1=g.findAt((4.025233, 0.75)),
    point1=(0.733189206279801, 0.182644009590149),
    curve2=g.findAt((1.95001, 0.720118)),
    point2=(1.62787567272877, 0.433001518249512))

s.breakCurve(curve1=g.findAt((4.822279, 0.898509)),
```

```
    point1=(2.34362044857835, 0.415118813514709),
    curve2=g.findAt((2.783134, 0.875349)),
    point2=(3.22040910681461, 0.701241612434387))

s.breakCurve(curve1=g.findAt((5.655403, 1.053741)),
    point1=(4.04351950354929, 0.772772312164307),
    curve2=g.findAt((4.558378, 1.090642)),
    point2=(5.02767302451382, 0.95159912109375))

s.breakCurve(curve1=g.findAt((4.025233, 1.25)),
    point1=(1.55629884485765, 1.09466052055359),
    curve2=g.findAt((1.95001, 0.720118)),
    point2=(2.32572264036564, 1.02312982082367))

s.breakCurve(curve1=g.findAt((5.178197, 1.321609)),
    point1=(3.79300780849022, 1.23772192001343),
    curve2=g.findAt((3.712795, 0.933089)),
    point2=(4.168774592922, 1.18407392501831))

s.VerticalDimension(vertex1=v.findAt((0.0, 1.0)),
    vertex2=v.findAt((0.0, 0.0)),
    textPoint=(-0.304643334343144, 0.0216999053955078), value=1.0)
s.HorizontalDimension(vertex1=v.findAt((0.0, 0.0)),
    vertex2=v.findAt((8.050466, 1.5)),
    textPoint=(6.26233672422377, -0.443249702453613), value=8.05046582503149)

s.ParallelConstraint(entity1=g.findAt((1.152964, 1.071609)), entity2=g.findAt((

    3.235589, 1.200957)))

s.ParallelConstraint(entity1=g.findAt((3.235589, 1.200957)), entity2=g.findAt((

    6.107858, 1.379348)))

s.ParallelConstraint(entity1=g.findAt((0.797047, 0.148509)), entity2=g.findAt((

    2.427216, 0.45225)))

s.ParallelConstraint(entity1=g.findAt((2.427216, 0.45225)), entity2=g.findAt((
    4.105923, 0.765035)))

s.ParallelConstraint(entity1=g.findAt((4.105923, 0.765035)), entity2=g.findAt((

    6.500986, 1.211294)))

s.EqualLengthConstraint(entity1=g.findAt((0.797047, 0.148509)),
    entity2=g.findAt((2.427216, 0.45225)))

s.EqualLengthConstraint(entity1=g.findAt((2.39114, 0.445528)),
    entity2=g.findAt((4.033769, 0.751591)))

s.EqualLengthConstraint(entity1=g.findAt((3.985233, 0.742547)),
    entity2=g.findAt((6.416373, 1.195528)))

s.EqualLengthConstraint(entity1=g.findAt((1.152964, 1.071609)),
    entity2=g.findAt((3.235589, 1.200957)))

s.EqualLengthConstraint(entity1=g.findAt((3.458891, 1.214826)),
    entity2=g.findAt((6.33116, 1.393217)))

mdb.models['standard'].ConstrainedSketch(name='Truss', objectToCopy=s)
```

```
p = mdb.models['standard'].parts['Truss']
d = p.datums
p.Wire(sketchPlane=d[4], sketchUpEdge=d[5], sketchPlaneSide=SIDE1,
    sketchOrientation=LEFT, sketch=s)
s.unsetPrimaryObject()
del mdb.models['standard'].sketches['__profile__']

session.viewports['Viewport: 1'].view.fitView()

p = mdb.models['standard'].parts['Truss']
d, r = p.datums, p.referencePoints
p.DatumPointByOffset(point=d[2], vector=(0.0, 0.0, 2.0))
p.DatumPointByOffset(point=r[1], vector=(0.0, 0.0, 2.0))
p.DatumPointByOffset(point=d[3], vector=(0.0, 0.0, 0.2))
p.DatumPlaneByThreePoints(point1=d[8], point2=d[9], point3=d[7])

t = p.MakeSketchTransform(sketchPlane=d[10], sketchUpEdge=d[5],
    sketchPlaneSide=SIDE1, sketchOrientation=LEFT, origin=(4.122203, 0.75,
    1.536252))
s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=683.38, gridSpacing=17.08, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.retrieveSketch(sketch=mdb.models['standard'].sketches['Truss'])
s.move(vector=(-4.14820681508444, -0.75), objectList=(g.findAt((0.0, 0.5)),
    g.findAt((1.006308, 0.6875)), g.findAt((2.348053, 0.770833)), g.findAt((
    3.354361, 0.958333)), g.findAt((4.696105, 1.041667)), g.findAt((5.702413,
    1.229167)), g.findAt((1.006308, 0.1875)), g.findAt((3.018925, 0.5625)),
    g.findAt((5.031541, 0.9375)), g.findAt((7.044158, 1.3125)), g.findAt((
    1.341744, 1.083333)), g.findAt((6.708722, 1.416667)), g.findAt((4.025233,
    1.25)), v.findAt((8.050466, 1.5)), v.findAt((0.0, 1.0))))
d = p.datums
p.Wire(sketchPlane=d[10], sketchUpEdge=d[5], sketchPlaneSide=SIDE1,
    sketchOrientation=LEFT, sketch=s)
s.unsetPrimaryObject()
del mdb.models['standard'].sketches['__profile__']

##
##  dummy part
##
p = mdb.models['standard'].Part(name='Truss-all',
    objectToCopy=mdb.models['standard'].parts['Truss'])
session.viewports['Viewport: 1'].setValues(displayedObject=p)
v = p.vertices
p.WirePolyLine(points=((v.findAt(coordinates=(2.0, 0.375, 1.775)), v.findAt(
    coordinates=(2.0, 0.375, 0.225))), (v.findAt(coordinates=(2.0, 0.375,
    0.225)), v.findAt(coordinates=(2.666667, 1.166667, 1.7))), (v.findAt(
    coordinates=(2.666667, 1.166667, 1.7)), v.findAt(coordinates=(2.666667,
    1.166667, 0.3))), (v.findAt(coordinates=(4.0, 0.75, 1.55)), v.findAt(
    coordinates=(4.0, 0.75, 0.45))), (v.findAt(coordinates=(4.0, 0.75, 0.45)),

    v.findAt(coordinates=(5.333333, 1.333333, 1.4))), (v.findAt(coordinates=(
    5.333333, 1.333333, 1.4)), v.findAt(coordinates=(5.333333, 1.333333,
    0.6))), (v.findAt(coordinates=(5.333333, 1.333333, 0.6)), v.findAt(
    coordinates=(6.0, 1.125, 1.325))), (v.findAt(coordinates=(6.0, 1.125,
    1.325)), v.findAt(coordinates=(6.0, 1.125, 0.675)))), mergeType=IMPRINT,
    meshable=ON)
##
```

```
##  create cross brace part in the assembly module
##

a = mdb.models['standard'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##  Set coordinate system (done by default)
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['standard'].parts['Truss']
a.Instance(name='Truss-1', part=p, dependent=ON)
p = mdb.models['standard'].parts['Truss-all']
a.Instance(name='Truss-all-1', part=p, dependent=ON)
a.PartFromBooleanCut(name='Cross brace',
    instanceToBeCut=a.instances['Truss-all-1'],
    cuttingInstances=(a.instances['Truss-1'], ))
p = mdb.models['standard'].parts['Cross brace']
a.Instance(name='Cross brace-1', part=p, dependent=ON)
a.suppressFeatures(('Truss-1', 'Truss-all-1', ))
a.features['Truss-1'].resume()

p = mdb.models['standard'].parts['Cross brace']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
##
##  Create beam profiles and beam sections
##
mdb.models['standard'].BoxProfile(name='MainBoxProfile', b=0.05,
    a=0.1, uniformThickness=ON, t1=0.005)
mdb.models['standard'].BoxProfile(name='BraceBoxProfile', b=0.03,
    a=0.03, uniformThickness=ON, t1=0.003)
mdb.models['standard'].BeamSection(name='MainMemberSection',
    profile='MainBoxProfile', poissonRatio=0.25,
    integration=BEFORE_ANALYSIS, table=((2.e11, 8.e10),))
mdb.models['standard'].sections['MainMemberSection'].setValues(density=7800.0)
mdb.models['standard'].BeamSection(name='BracingSection',
    profile='BraceBoxProfile', poissonRatio=0.25,
    integration=BEFORE_ANALYSIS, table=((2.e11, 8.e10),))
mdb.models['standard'].sections['BracingSection'].setValues(density=7800.0)
##
##  Flip tangent directions
##
##p = mdb.models['standard'].parts['Truss']
##e = p.edges
##edges = e
##regions = regionToolset.Region(edges=edges)
##p.flipTangent(regions=regions)
####
##edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((2.5, 0.46875, 0.28125), ),
((
##    0.5, 0.09375, 0.05625), ), ((4.5, 0.84375, 1.49375), ), ((2.5, 0.46875,
##    1.71875), ), ((0.5, 0.09375, 1.94375), ))
##regions = regionToolset.Region(edges=edges)
##p.flipTangent(regions=regions)
##
##  Assign beam sections to the cross brace
##
p = mdb.models['standard'].parts['Cross brace']
e = p.edges
edges = e.findAt(((2.666667, 1.166667, 1.35), ),
    ((2.166667, 0.572917, 0.59375), ), ((2.0, 0.375, 1.3875), ),
    ((6.0, 1.125, 1.1625), ), ((5.5, 1.28125, 0.78125), ),
    ((5.333333, 1.333333, 1.2), ), ((4.333333, 0.895833, 0.6875), ),
    ((4.0, 0.75, 1.275), ))
```

```
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='BracingSection')
##
##  Assign beam sections to the internal bracing
##
p = mdb.models['standard'].parts['Truss']
e = p.edges
edges = e.findAt(((3.0, 1.0625, 0.3375), ), ((5.5, 1.28125, 0.61875), ),
    ((4.333333, 0.895833, 0.4875), ), ((0.5, 0.84375, 0.05625), ),
    ((2.166667, 0.572917, 0.24375), ), ((3.0, 1.0625, 1.6625), ),
    ((5.5, 1.28125, 1.38125), ), ((4.333333, 0.895833, 1.5125), ),
    ((0.5, 0.84375, 1.94375), ), ((2.166667, 0.572917, 1.75625), ))
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='BracingSection')
##
##  Assign beam sections to the main members
##
p = mdb.models['standard'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((4.666667, 1.291667, 0.525), ),

    ((6.5, 1.21875, 0.73125), ), ((7.333333, 1.458333, 0.825), ),
    ((2.5, 0.46875, 0.28125), ), ((2.0, 1.125, 0.225), ),
    ((0.5, 0.09375, 0.05625), ), ((4.5, 0.84375, 1.49375), ),
    ((4.666667, 1.291667, 1.475), ), ((6.5, 1.21875, 1.26875), ),
    ((7.333333, 1.458333, 1.175), ), ((2.5, 0.46875, 1.71875), ),
    ((2.0, 1.125, 1.775), ), ((0.5, 0.09375, 1.94375), ))
region = regionToolset.Region(edges=edges)
p.SectionAssignment(region=region, sectionName='MainMemberSection')
##
##  Assign beam section orientations to truss 'B'
##
p = mdb.models['standard'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 0.50625), ), ((3.0, 1.0625, 0.3375), ),
    ((4.666667, 1.291667, 0.525), ), ((5.5, 1.28125, 0.61875), ),
    ((6.5, 1.21875, 0.73125), ), ((7.333333, 1.458333, 0.825), ),
    ((4.333333, 0.895833, 0.4875), ), ((2.5, 0.46875, 0.28125), ),
    ((0.5, 0.84375, 0.05625), ), ((2.0, 1.125, 0.225), ),
    ((2.166667, 0.572917, 0.24375), ), ((0.5, 0.09375, 0.05625), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(-0.1118,
    0.0, 0.9936))
##
##  Assign beam section orientations to truss 'A'
##
p = mdb.models['standard'].parts['Truss']
e = p.edges
edges = e.findAt(((4.5, 0.84375, 1.49375), ), ((3.0, 1.0625, 1.6625), ),
    ((4.666667, 1.291667, 1.475), ), ((5.5, 1.28125, 1.38125), ),
    ((6.5, 1.21875, 1.26875), ), ((7.333333, 1.458333, 1.175), ),
    ((4.333333, 0.895833, 1.5125), ), ((2.5, 0.46875, 1.71875), ),
    ((0.5, 0.84375, 1.94375), ), ((2.0, 1.125, 1.775), ),
    ((2.166667, 0.572917, 1.75625), ), ((0.5, 0.09375, 1.94375), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(-0.1118,
    0.0, -0.9936))
##
##  Assign beam section orientations to 'Cross brace'
##
p = mdb.models['standard'].parts['Cross brace']
```

```
e = p.edges
edges = e.findAt(((2.666667, 1.166667, 1.35), ), ((2.166667, 0.572917,
    0.59375), ), ((2.0, 0.375, 1.3875), ), ((6.0, 1.125, 1.1625), ), ((5.5,
    1.28125, 0.78125), ), ((5.333333, 1.333333, 1.2), ), ((4.333333, 0.895833,

    0.6875), ), ((4.0, 0.75, 1.275), ))
region=regionToolset.Region(edges=edges)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(0.0, 1.0, 0.0))

session.viewports['Viewport: 1'].setValues(displayedObject=a)
a.regenerate()
##
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', 'Cross brace-1'))
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
   geometryEdgesInShaded=OFF, datumPoints=OFF, datumAxes=OFF, datumPlanes=OFF,

   datumCoordSystems=OFF, referencePointLabels=OFF, referencePointSymbols=OFF)
##
##  Translate the truss
##
p = a.instances['Truss-1']
p.translate(vector=(0.0, -0.5, -1.0))
##
##  Translate the cross brace
##
p = a.instances['Cross brace-1']
p.translate(vector=(0.0, -0.5, -1.0))


##
##  Create geometry sets
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', ))
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
    datumPoints=OFF)

v = a.instances['Truss-1'].vertices
verts = v.findAt(((0.0, 0.5, -1.0), ), ((0.0, -0.5, -1.0), ), ((0.0, 0.5,
    1.0), ), ((0.0, -0.5, 1.0), ))
a.Set(vertices=verts, name='Attach')

verts = v.findAt(((8.0, 1.0, 0.1), ))
a.Set(vertices=verts, name='Tip-a')

verts = v.findAt(((8.0, 1.0, -0.1), ))
a.Set(vertices=verts, name='Tip-b')

e = a.instances['Truss-1'].edges
edges = e.findAt(((6.5, 0.71875, 0.26875), ))
a.Set(edges=edges, name='Leg-a')

edges = e.findAt(((6.5, 0.71875, -0.26875), ))
a.Set(edges=edges, name='Leg-b')

edges = e.findAt(((4.333333, 0.395833, 0.5125), ))
a.Set(edges=edges, name='Inner-a')
```

```
edges = e.findAt(((4.333333, 0.395833, -0.5125), ))
a.Set(edges=edges, name='Inner-b')

session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Cross brace-1', ))
session.viewports['Viewport: 1'].view.setValues(
    cameraPosition=(7.4736, 8.3559, 20.002),
    cameraUpVector=(-0.34023, 0.72044, -0.60432))

session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    visibleInstances=('Truss-1', 'Cross brace-1'))
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])

##
##  Create a frequency extraction step
##
mdb.models['standard'].FrequencyStep(name='Extract frequencies',
    previous='Initial',
    description='First 30 modes', numEigen=30,
    maxEigen=None, shift=None, eigensolver=LANCZOS, minEigen=None,
    blockSize=DEFAULT, maxBlocks=DEFAULT)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Extract frequencies')
##
##  Create a modal dynamics step
##
mdb.models['standard'].ModalDynamicsStep(name='Transient modal dynamics',
    previous='Extract frequencies',
    description='Crane Response to Dropped Load',
    continueAnalysis=OFF, timePeriod=0.5, incSize=0.005, directDamping=((1,
    30, 0.05), ), compositeDamping=None, rayleighDamping=None, amplitude=STEP)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Transient modal dynamics')
##
##  Modify output requests
##
mdb.models['standard'].fieldOutputRequests['F-Output-2'].setValues(
    variables=('U', ), frequency=5)
mdb.models['standard'].historyOutputRequests['H-Output-2'].setValues(
    variables=('ALLIE', 'ALLKE', 'ALLVD'), frequency=1)
a0=mdb.models['standard'].rootAssembly
regionDef=a0.sets['Tip-a']
mdb.models['standard'].HistoryOutputRequest(name='H-Output-3',
    createStepName='Transient modal dynamics',
    variables=('U1', 'U2', 'U3'),
    region=regionDef, frequency=1)
a0=mdb.models['standard'].rootAssembly
regionDef=a0.sets['Attach']
mdb.models['standard'].HistoryOutputRequest(name='H-Output-4',
    createStepName='Transient modal dynamics',
    variables=('RF1', 'RF2', 'RF3'),
    region=regionDef, frequency=1)

##
##  Create constraints between the tips of the truss
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(interactions=ON)
mdb.models['standard'].Equation(name='TipConstraint-1',
    terms=((1.0, 'Tip-a', 1), (-1.0, 'Tip-b', 1)))
mdb.models['standard'].Constraint('TipConstraint-2',
    mdb.models['standard'].constraints['TipConstraint-1'])
```

```
mdb.models['standard'].constraints['TipConstraint-2'].setValues(
    terms=((1.0, 'Tip-a', 2), (-1.0, 'Tip-b', 2)))
##
##  Create JOIN connectors between truss and bracing
##
v1 = a.instances['Cross brace-1'].vertices
v2 = a.instances['Truss-1'].vertices

a.WirePolyLine(points=((v1.findAt(coordinates=(2.0, -0.125, 0.775)), v2.findAt(

    coordinates=(2.0, -0.125, 0.775))), (v1.findAt(coordinates=(2.666667,
    0.666667, 0.7)), v2.findAt(coordinates=(2.666667, 0.666667, 0.7))), (
    v1.findAt(coordinates=(2.0, -0.125, -0.775)), v2.findAt(coordinates=(2.0,
    -0.125, -0.775))), (v1.findAt(coordinates=(2.666667, 0.666667, -0.7)),
    v2.findAt(coordinates=(2.666667, 0.666667, -0.7))), (v1.findAt(
    coordinates=(4.0, 0.25, -0.55)), v2.findAt(coordinates=(4.0, 0.25,
    -0.55))), (v1.findAt(coordinates=(4.0, 0.25, 0.55)), v2.findAt(
    coordinates=(4.0, 0.25, 0.55))), (v1.findAt(coordinates=(5.333333,
    0.833333, 0.4)), v2.findAt(coordinates=(5.333333, 0.833333, 0.4))), (
    v1.findAt(coordinates=(5.333333, 0.833333, -0.4)), v2.findAt(coordinates=(
    5.333333, 0.833333, -0.4))), (v1.findAt(coordinates=(6.0, 0.625, -0.325)),

    v2.findAt(coordinates=(6.0, 0.625, -0.325))), (v1.findAt(coordinates=(6.0,

    0.625, 0.325)), v2.findAt(coordinates=(6.0, 0.625, 0.325)))),
    mergeType=IMPRINT, meshable=OFF)

e1 = a.edges
edges1 = e1.findAt(((6.000025, 0.625, 0.325), ), ((6.000025, 0.625, -0.325),
),
    ((5.333358, 0.833333, -0.4), ), ((5.333358, 0.833333, 0.4), ), ((4.000025,

    0.25, 0.55), ), ((4.000025, 0.25, -0.55), ), ((2.666692, 0.666667, -0.7),
    ), ((2.000025, -0.125, -0.775), ), ((2.666692, 0.666667, 0.7), ), ((
    2.000025, -0.125, 0.775), ))
a.Set(edges=edges1, name='Wire-1-Edge-1')

mdb.models['standard'].ConnectorSection(name='ConnSect-1',
    translationalType=JOIN)

region=a.sets['Wire-1-Edge-1']
a.SectionAssignment(sectionName='ConnSect-1', region=region)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON, interactions=OFF)
##
##  Apply encastre bc to set 'Attach'
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Extract frequencies')
a = mdb.models['standard'].rootAssembly
region = a.sets['Attach']
mdb.models['standard'].EncastreBC(name='Fixed end',
    createStepName='Extract frequencies', region=region)
##
##  Create amplitude curve 'Bounce'
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Transient modal dynamics')
mdb.models['standard'].TabularAmplitude(name='Bounce', timeSpan=STEP,
    smooth=0.25, data=((0.0, 0.0), (0.01, 1.0), (0.2, 1.0), (0.21, 0.0)))
```

```
##
##  Apply concentrated force to set 'Tip-b'
##
a = mdb.models['standard'].rootAssembly
region = a.sets['Tip-b']
mdb.models['standard'].ConcentratedForce(name='Tip load',
    createStepName='Transient modal dynamics', region=region, cf2=-10000.0,
    amplitude='Bounce')
##
##  Assign global seed
##
p = mdb.models['standard'].parts['Truss']
p.seedPart(size=2.0)
p = mdb.models['standard'].parts['Cross brace']
p.seedPart(size=2.0)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=B33)

p = mdb.models['standard'].parts['Cross brace']
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))

p = mdb.models['standard'].parts['Truss']
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))
##
##  Generate mesh
##
p = mdb.models['standard'].parts['Truss']
p.generateMesh()
p = mdb.models['standard'].parts['Cross brace']
p.generateMesh()

##
##  Add keywords
##
mdb.models['standard'].keywordBlock.synchVersions()
mdb.models['standard'].keywordBlock.insert(GetBlockPosition('standard', '*End
Assembly')-1, """*NORMAL, TYPE=ELEMENT
Inner-a,  Inner-a, -0.3962,  0.9171,  0.0446
Inner-b,  Inner-b,  0.3962, -0.9171,  0.0446
Leg-a,    Leg-a,   -0.1820,  0.9829,  0.0205
Leg-b,    Leg-b,    0.1820, -0.9829,  0.0205""")
##
##  Create job
##
mdb.Job(name='DynCrane', model='standard',
    description='3-D model of light-service cargo crane')

a = mdb.models['standard'].rootAssembly
a.regenerate()
##
##  Save model database
##
mdb.saveAs('DynCrane.cae')
```

```
mdb.Model(name='explicit', objectToCopy=mdb.models['standard'])

a = mdb.models['explicit'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  delete modal dynamics step
##
del mdb.models['explicit'].steps['Transient modal dynamics']
##
##  replace frequency extraction step with explicit dynamics step
##
mdb.models['explicit'].ExplicitDynamicsStep(name='Extract frequencies',
    previous='Initial', maintainAttributes=True, timePeriod=0.5, nlgeom=OFF)
##
##  rename explicit dynamics step
##
mdb.models['explicit'].steps.changeKey(fromName='Extract frequencies',
    toName='Transient dynamics')
##
##  Redefine history output
##
regionDef=mdb.models['explicit'].rootAssembly.sets['Tip-a']
mdb.models['explicit'].HistoryOutputRequest(name='H-Output-3',
    createStepName='Transient dynamics', variables=('U1', 'U2', 'U3'),
    region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)

regionDef=mdb.models['explicit'].rootAssembly.sets['Attach']
mdb.models['explicit'].HistoryOutputRequest(name='H-Output-4',
    createStepName='Transient dynamics', variables=('RF1', 'RF2', 'RF3'),
    region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)
##
##  Add mass proportional damping
##
mdb.models['explicit'].sections['BracingSection'].setValues(poissonRatio=0.0,
    density=7800.0, thermalExpansion=OFF, temperatureDependency=OFF,
    dependencies=0, table=((2.e11, 8.e10), ),
    alphaDamping=15.0, betaDamping=0.0, compositeDamping=0.0, centroid=(0.0,
    0.0), shearCenter=(0.0, 0.0))

mdb.models['explicit'].sections['MainMemberSection'].setValues(
    poissonRatio=0.0, density=7800.0, thermalExpansion=OFF,
    temperatureDependency=OFF, dependencies=0, table=((2.e11,
    8.e10), ), alphaDamping=15.0, betaDamping=0.0,
    compositeDamping=0.0, centroid=(0.0, 0.0), shearCenter=(0.0, 0.0))
##
##  Redefine tip load
##
a = mdb.models['explicit'].rootAssembly
region = a.sets['Tip-b']
mdb.models['explicit'].ConcentratedForce(name='Load-1',
    createStepName='Transient dynamics', region=region, cf2=-10000.0,
    amplitude='Bounce', localCsys=None)
##
##  Change element library
##
elemType1 = mesh.ElemType(elemCode=B31, elemLibrary=EXPLICIT)

p = mdb.models['explicit'].parts['Cross brace']
e = p.edges
edges = e
```

```
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))

p = mdb.models['explicit'].parts['Truss']
elemType1 = mesh.ElemType(elemCode=B31, elemLibrary=EXPLICIT)
e = p.edges
edges = e
pickedRegions =(edges, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, ))

a = mdb.models['explicit'].rootAssembly
a.regenerate()
##
##  Create explicit job
##
mdb.Job(name='expDynCrane', model='explicit', type=ANALYSIS)

a = mdb.models['standard'].rootAssembly
a.regenerate()
a = mdb.models['explicit'].rootAssembly
a.regenerate()

session.viewports['Viewport: 1'].setValues(displayedObject=a)

mdb.save()
```

Nonlinear skew plate

This Python script can be used to create a model for examples contained in this guide.

- gsi_nlskewplate_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_nlskewplate_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for nonlinear skew plate example
#

from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

##
##  Sketch profile of the plate
##
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=4.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-0.275, 0.25), point2=(0.575, -0.25))
s.delete(objectList=(c[18], c[19], c[20], c[21], c[22]))
s.VerticalConstraint(entity=g.findAt((-0.275, 0.0)))
s.VerticalConstraint(entity=g.findAt((0.575, 0.0)))
s.ParallelConstraint(entity1=g.findAt((0.15, -0.25)), entity2=g.findAt((0.15,
    0.25)))
s.FixedConstraint(entity=g.findAt((0.15, -0.25)))
s.ObliqueDimension(vertex1=v.findAt((-0.275, 0.25)), vertex2=v.findAt((-0.275,

    -0.25)), textPoint=(-0.384168207645416, -0.039074968546629), value=0.4)
```

```
s.delete(objectList=(c[26], ))
s.HorizontalDimension(vertex1=v.findAt((-0.275, -0.25)), vertex2=v.findAt((
    0.575, -0.25)), textPoint=(0.502545475959778, -0.405263155698776),
    value=1.0)
s.AngularDimension(line1=g.findAt((-0.275, -0.05)), line2=g.findAt((0.225,
    -0.25)), textPoint=(-0.221120104193687, -0.173046261072159), value=60.0)
session.viewports['Viewport: 1'].view.fitView()
s.move(vector=(-0.725, 0.05), objectList=(g.findAt((-0.275, -0.05)), g.findAt((

    0.225, 0.038675)), g.findAt((0.725, 0.52735)), g.findAt((0.225,
    0.438675))))
p = mdb.models['Model-1'].Part(name='Plate', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Plate']
p.BaseShell(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Plate']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

##
##  Create material 'Metal'
##
mdb.models['Model-1'].Material('Metal')
mdb.models['Model-1'].materials['Metal'].Elastic(table=((30.0E9, 0.3), ))
##
##  Create shell section
##
mdb.models['Model-1'].HomogeneousShellSection(name='PlateSection',
    preIntegrate=ON,
    material='Metal', thickness=0.008,
    poissonDefinition=DEFAULT, temperature=GRADIENT)
##
##  Create datum coordinate system
##
p1 = mdb.models['Model-1'].parts['Plate']
e = p1.edges
p1.DatumCsysByTwoLines(CARTESIAN, line1=e.findAt((-0.75,
    -0.0556624327025937, 0.0), ), line2=e.findAt((0.0,
    0.477350269189625, 0.0), ))
##
##  Assign material orientation to the plate
##
p0 = mdb.models['Model-1'].parts['Plate']
f = p0.faces
faces = f[0:1]
region=(None, None,
    f.findAt(((-0.666666666666667,
    0.125783423063208, 0.0), (0.0, 0.0, 1.0)), ), None)
datum = p0.datums[2]
p0.MaterialOrientation(region=region, localCsys=datum, axis=AXIS_3)
##
##  Assign section to the plate
##
p0 = mdb.models['Model-1'].parts['Plate']
f = p0.faces
faces = f[0:1]
region=(None, None,
    f.findAt(((-0.666666666666667,
    0.125783423063208, 0.0), (0.0, 0.0, 1.0)), ), None)
p0 = mdb.models['Model-1'].parts['Plate']
```

```
p0.SectionAssignment(region=region, sectionName='PlateSection')

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the plate
##
p = mdb.models['Model-1'].parts['Plate']
a.Instance(name='Plate-1', part=p, dependent=ON)
##
##  Partition the plate
##
p = mdb.models['Model-1'].parts['Plate']
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

faces=(f.findAt((
    -0.666666666666667, 0.125783423063208, 0.0), (0.0, 0.0, 1.0)), )

p.PartitionFaceByShortestPath(faces=faces,
    point1=p.InterestingPoint(e.findAt((-0.75,
    -0.0556624327025937, 0.0), ), MIDDLE),
    point2=p.InterestingPoint(e.findAt((-0.25,
    0.63301270189222, 0.0), ), MIDDLE))

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums
##
##  Create geometry set 'MidSpan'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (-0.5, 0.188675134594813, 0.0), ), ), name='MidSpan')
##
##  Create geometry set 'EndA'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (-1.0, 0.1, 0.0), ), ), name='EndA')
##
##  Create geometry set 'EndB'
##
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Plate-1'].edges
a.Set(edges=e1.findAt((
    (0.0, 0.477350269189625, 0.0), ), ), name='EndB')

##
##  Create a static general step
##
mdb.models['Model-1'].StaticStep(name='Apply pressure',
    previous='Initial',
    description='Nonlinear analysis: Uniform pressure (20 kPa) load',
    timePeriod=1, adiabatic=OFF, maxNumInc=100,
    stabilization=None, timeIncrementationMethod=AUTOMATIC, initialInc=0.1,
    minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT, amplitude=RAMP,
```

```
    extrapolation=LINEAR, fullyPlastic="", nlgeom=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Apply pressure')
##
##  Modify output requests
##
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=PRESELECT, frequency=2)
a0=mdb.models['Model-1'].rootAssembly
regionDef=a0.sets['MidSpan']
mdb.models['Model-1'].historyOutputRequests['H-Output-1'].setValues(
    variables=('U3',), region=regionDef)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
##
##  Apply bc in local CSYS to set EndB (right edge)
##
a0 = mdb.models['Model-1'].rootAssembly
region = a0.sets['EndB']
datum = mdb.models['Model-1'].rootAssembly.instances['Plate-1'].datums[2]
mdb.models['Model-1'].DisplacementBC(name='Rail boundary condition',
    createStepName='Apply pressure', region=region, u2=0.0, u3=0.0, ur1=0.0,
    ur2=0.0, ur3=0.0, localCsys=datum)
##
##  Apply encastre bc to set EndA (left edge)
##
a0 = mdb.models['Model-1'].rootAssembly
region = a0.sets['EndA']
mdb.models['Model-1'].EncastreBC(name='Fix left end',
    createStepName='Apply pressure', region=region)
##
##  Apply pressure load
##
a0 = mdb.models['Model-1'].rootAssembly
f1 = a0.instances['Plate-1'].faces
region=((
    f1.findAt(((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)), ((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)), ), SIDE1),
    )
mdb.models['Model-1'].Pressure(name='Pressure',
    createStepName='Apply pressure', region=region, magnitude=2.0E4)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON, loads=OFF,
    bcs=OFF, predefinedFields=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
##
##  Assign global seed
##
p.seedPart(size=0.1)
##
##  Use structured meshing
##
f1 = p.faces
regions=(
    f1.findAt((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)),
    f1.findAt((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)))
```

```
p.setMeshControls(regions=regions, technique=STRUCTURED)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=S8R5)
elemType2 = mesh.ElemType(elemCode=STRI65)
regions=(None, None,
    f1.findAt(((
    -0.666666666666667, 0.259116756396542, 0.0), (0.0, 0.0, 1.0)), ((
    -0.333333333333333, 0.318233512793084, 0.0), (0.0, 0.0, 1.0)), ), None)
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2))
##
##  Generate mesh
##
p.generateMesh()

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
##
##  Create job
##
mdb.Job(name='NlSkewPlate', model='Model-1',
    description='Nonlinear Elastic Skew Plate. 20 kPa Load.')

a = mdb.models['Model-1'].rootAssembly
a.regenerate()

##
##  Save model database
##
mdb.saveAs('NlSkewPlate.cae')
```

Stress wave propagation in a bar

This Python script can be used to create a model for examples contained in this guide.

- gxi_stresswave_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gxi_stresswave_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Replay file for stress wave in a bar example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
Mdb()


s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=1.0)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=1.0, gridSpacing=0.02, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.02,
decimalPlaces=2)
s.rectangle(point1=(-0.1, -0.1), point2=(0.1, 0.1))
p = mdb.models['Model-1'].Part(name='Bar', dimensionality=THREE_D,
type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Bar']
p.BaseSolidExtrude(sketch=s, depth=1.0)
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

session.viewports['Viewport: 1'].partDisplay.setValues(renderStyle=SHADED)

mdb.models['Model-1'].Material('Steel')
mdb.models['Model-1'].materials['Steel'].Density(table=((7800.0, ), ))
mdb.models['Model-1'].materials['Steel'].Elastic(table=((207.0E9, 0.3), ))
```

```
mdb.models['Model-1'].HomogeneousSolidSection(name='BarSection',
    material='Steel', thickness=1.0)

c = p.cells
cells = c
region =(None, None, None, cells)
p.SectionAssignment(region=region, sectionName='BarSection')

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

a.DatumCsysByDefault(CARTESIAN)
a.Instance(name='Bar-1', part=p, dependent=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(renderStyle=SHADED)

e = a.instances['Bar-1'].edges
edges = e.findAt(((0.1, 0.1, 0.25), ))
a.Set(edges=edges, name='out')

f = a.instances['Bar-1'].faces
faces = f.findAt(((-0.1, 0.0333333333333333, 0.666666666666667), ))
a.Set(faces=faces, name='back')

faces = f.findAt(((0.1, -0.0333333333333333, 0.666666666666667), ))
a.Set(faces=faces, name='front')

faces = f.findAt(((-0.0333333333333333, -0.1, 0.666666666666667), ))
a.Set(faces=faces, name='bot')

faces = f.findAt(((0.0333333333333333, 0.1, 0.666666666666667), ))
a.Set(faces=faces, name='top')

faces = f.findAt(((0.0333333333333333, -0.0333333333333333, 0.0), ))
a.Set(faces=faces, name='fix')

side1Faces1 = f.findAt(((-0.0333333333333333, -0.0333333333333333, 1.0), ))
a.Surface(side1Faces=side1Faces1, name='load')

mdb.models['Model-1'].ExplicitDynamicsStep(name='BlastLoad', previous='Initial',

    description='Apply pressure load pulse', timePeriod=0.0002,
    adiabatic=OFF, timeIncrementationMethod=AUTOMATIC_GLOBAL, scaleFactor=1,
    maxIncrement=None, massScaling=PREVIOUS_STEP, linearBulkViscosity=0.06,
    quadBulkViscosity=0.0)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='BlastLoad')

mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    numIntervals=4)
del mdb.models['Model-1'].historyOutputRequests['H-Output-1']
regionDef=mdb.models['Model-1'].rootAssembly.sets['out']
mdb.models['Model-1'].HistoryOutputRequest(name='H-Output-1',
    createStepName='BlastLoad', variables=('S33', ), frequency=1,
    region=regionDef)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])

region = a.sets['fix']
mdb.models['Model-1'].DisplacementBC(name='Fix right end',
```

```
createStepName='Initial',
    region=region, u1=0.0, u2=0.0, u3=0.0)

region = a.sets['back']
mdb.models['Model-1'].DisplacementBC(name='Restrain back face',
createStepName='Initial',
    region=region, u1=0.0)

region = a.sets['front']
mdb.models['Model-1'].DisplacementBC(name='Restrain front face',
createStepName='Initial',
    region=region, u1=0.0)

region = a.sets['bot']
mdb.models['Model-1'].DisplacementBC(name='Restrain bottom face',
createStepName='Initial',
    region=region, u2=0.0)

region = a.sets['top']
mdb.models['Model-1'].DisplacementBC(name='Restrain top face',
createStepName='Initial',
    region=region, u2=0.0)

mdb.models['Model-1'].TabularAmplitude(name='Blast', timeSpan=STEP,
    smooth=0.25, data=((0.0, 1.0), (3.88e-05, 1.0), (3.89e-05, 0.0), (3.9e-05,
     0.0)))

region = a.surfaces['load']
mdb.models['Model-1'].Pressure(name='Blast load', createStepName='BlastLoad',
    region=region, magnitude=1.0E5, amplitude='Blast')

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON, loads=OFF,
    bcs=OFF, predefinedFields=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)

p = mdb.models['Model-1'].parts['Bar']
p.seedPart(size=0.02)
import re, uti
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=0.04)

elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
    kinematicSplit=AVERAGE_STRAIN, hourglassControl=RELAX_STIFFNESS)
elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
c = p.cells
cells = c
regions =(None, None, None, cells)
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2, elemType3))
p.generateMesh()

mdb.Job(name='Bar', model='Model-1', description='Stress wave propagation in a
 bar (SI units)')

a.regenerate()

mdb.saveAs('Bar')
```

Connecting lug with plasticity

This Python script can be used to create a model for examples contained in this guide.

- gsi_plasticlug_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_plasticlug_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for plastic lug example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

session.viewports['Viewport: 1'].setValues(displayedObject=None)

##  Sketch profile of the lug

s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=0.25)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-0.0275, 0.0225), point2=(0.02125, -0.01375))
s.delete(objectList=(g.findAt((0.02125, 0.004375)), ))
s.EqualLengthConstraint(entity1=g.findAt((-0.003125, -0.01375)),
    entity2=g.findAt((-0.003125, 0.0225)))
s.ObliqueDimension(vertex1=v.findAt((-0.0275, -0.01375)), vertex2=v.findAt((
    0.02125, -0.01375)), textPoint=(-0.010330107063055, -0.023654306307435),
    value=0.1)
s.move(vector=(-0.025625, 0.0), objectList=(g.findAt((-0.0275, 0.004375)),
```

```
    g.findAt((0.0225, -0.01375)), g.findAt((0.0225, 0.0225))))
session.viewports['Viewport: 1'].view.fitView()
s.ObliqueDimension(vertex1=v.findAt((-0.053125, 0.0225)), vertex2=v.findAt((
    -0.053125, -0.01375)), textPoint=(-0.0564765408635139,
    0.00896133482456207), value=0.05)
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(nearPlane=0.230597,
    farPlane=0.279546, width=0.144889, height=0.121934)
s.Arc3Points(point1=(0.046875, 0.03625), point2=(0.046875, -0.01375), point3=(
    0.0625, 0.02375))
s.TangentConstraint(entity1=g.findAt((-0.003125, 0.03625)), entity2=g.findAt((
    0.0657, 0.01125)))
s.move(vector=(-0.001797, 0.0), objectList=(g.findAt((-0.053125, 0.01125)),
   g.findAt((-0.003125, -0.01375)), g.findAt((-0.003125, 0.03625)), g.findAt((

    0.071875, 0.01125))))
s.TangentConstraint(entity1=g.findAt((-0.004922, -0.01375)), entity2=g.findAt((

    0.070078, 0.01125)))
session.viewports['Viewport: 1'].view.fitView()
s.RadialDimension(curve=g.findAt((0.070078, 0.01125)), textPoint=(
    0.0655877739191055, -0.0193991288542748), radius=0.025)
d[2].setValues(reference=ON)
s.CircleByCenterPerimeter(center=(0.0425, 0.0125), point1=(0.0525, 0.01625))
s.ConcentricConstraint(entity1=g.findAt((0.0325, 0.00875)), entity2=g.findAt((
    0.070078, 0.01125)))
s.RadialDimension(curve=g.findAt((0.035078, 0.0075)), textPoint=(
    0.0297759883105755, 0.0219836011528969), radius=0.015)
s.VerticalDimension(vertex1=v.findAt((0.045078, 0.01125)), vertex2=v.findAt((
    0.059123, 0.016517)), textPoint=(0.0632773488759995, 0.0129371425136924),
    value=0.0)
s.VerticalDimension(vertex1=v.findAt((0.045078, 0.01125)), vertex2=v.findAt((
    0.067003, 0.023263)), textPoint=(0.0746369957923889, 0.012744665145874),
    value=0.0)
s.move(vector=(0.034922, 0.00375), objectList=(g.findAt((-0.054922, 0.01125)),

    g.findAt((-0.004922, -0.01375)), g.findAt((-0.004922, 0.03625)),
    g.findAt((0.070078, 0.01125)), g.findAt((0.030078, 0.01125))))
p = mdb.models['Model-1'].Part(name='Lug', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Lug']
p.BaseSolidExtrude(sketch=s, depth=0.02)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Lug']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Create material 'Steel'
##
mdb.models['Model-1'].Material('Steel')
mdb.models['Model-1'].materials['Steel'].Elastic(table=((200.0E9, 0.3), ))
mdb.models['Model-1'].materials['Steel'].Plastic(table=((380.0E6, 0.0), (
    580.0E6, 0.35)))
##
##  Create solid section
##
mdb.models['Model-1'].HomogeneousSolidSection(name='LugSection',
    material='Steel', thickness=1.0)
##
```

```
##  Assign solid section
##
c = p.cells
cells = c
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='LugSection', offset=0.0)
##
##  Set coordinate system (done by default)
##
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the lug
##
a.Instance(name='Lug-1', part=p, dependent=ON)
##
##  Create a static general step
##
mdb.models['Model-1'].StaticStep(name='LugLoad',
    previous='Initial', description='Apply uniform pressure to the hole',
    initialInc=0.2)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='LugLoad')
##
##  Apply encastre bc to left end
##
f = a.instances['Lug-1'].faces
faces = f.findAt(((-0.02, 0.006667, 0.013333), ))
region = regionToolset.Region(faces=faces)
mdb.models['Model-1'].EncastreBC(name='Fix left end',
    createStepName='LugLoad', region=region)

session.viewports['Viewport: 1'].setValues(displayedObject=a)
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
##
##  Partition the lug
##
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

pickedCells = c.findAt(((-0.02, 0.006667, 0.013333), ))
p.PartitionCellByPlaneThreePoints(point1=v.findAt(coordinates=(0.095, 0.015,
    0.02)), cells=pickedCells, point2=p.InterestingPoint(edge=e.findAt(
    coordinates=(0.08, 0.03, 0.02)), rule=MIDDLE), point3=p.InterestingPoint(
    edge=e.findAt(coordinates=(0.08, 0.03, 0.0)), rule=MIDDLE))
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Apply pressure load to bottom of hole
##
a.regenerate()
s = a.instances['Lug-1'].faces
side1Faces = s.findAt(((0.094968, 0.01402, 0.013333), ))
region = regionToolset.Region(side1Faces=side1Faces)
mdb.models['Model-1'].Pressure(name='Pressure load',
    createStepName='LugLoad', region=region, magnitude=1.0E8)


##
##  Create partitions
##
pickedCells = c.findAt(((-0.02, 0.006667, 0.006667), ), ((-0.02, 0.023333,
```

```
    0.013333), ))
p.PartitionCellByPlaneThreePoints(cells=pickedCells, point1=p.InterestingPoint(

    edge=e.findAt(coordinates=(0.069393, 0.004393, 0.0)), rule=MIDDLE),
    point2=p.InterestingPoint(edge=e.findAt(coordinates=(0.069393, 0.004393,
    0.02)), rule=MIDDLE), point3=p.InterestingPoint(edge=e.findAt(
    coordinates=(0.090607, 0.025607, 0.02)), rule=MIDDLE))

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

p.DatumPointByEdgeParam(edge=e.findAt(coordinates=(0.055, -0.01, 0.02)),
    parameter=0.25)
pickedCells = c.findAt(((-0.02, 0.006667, 0.006667), ), ((-0.02, 0.023333,
    0.013333), ))
p.PartitionCellByPlanePointNormal(point=d[5], normal=e.findAt(coordinates=(
    0.055, -0.01, 0.02)), cells=pickedCells)
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums


f = a.instances['Lug-1'].faces
faces = f.findAt(((-0.02, 0.006667, 0.006667), ), ((-0.02, 0.023333,
    0.013333), ))
a.Set(faces=faces, name='BuiltIn')

e = a.instances['Lug-1'].edges
edges = e.findAt(((0.08, 0.0, 0.015), ))
a.Set(edges=edges, name='HoleBot')

##
##  Assign global seed
##
p.seedPart(size=0.007)
import re, uti
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=0.01)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=C3D20R)
elemType2 = mesh.ElemType(elemCode=C3D15)
elemType3 = mesh.ElemType(elemCode=C3D10M)
cells = c
pickedRegions =(cells, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))
##
##  Generate the mesh
##
p.generateMesh()
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
    datumPoints=OFF)

mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=PRESELECT)
del mdb.models['Model-1'].historyOutputRequests['H-Output-1']

a.regenerate()
mdb.Job(name='PlasticLugHard', model='Model-1',
    description='Elastic-Plastic Steel Connecting Lug')
```

```
a.regenerate()

session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
#
#  Save model database
#
mdb.saveAs('PlasticLug.cae')
```

Blast loading on a stiffened plate

This Python script can be used to create a model for examples contained in this guide.

- gxi_stiffplate_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gxi_stiffplate_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for stiffened plate example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=5.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.Line(point1=(-0.65, 0.075), point2=(0.725000000046566, 0.075))
s.HorizontalConstraint(entity=g.findAt((0.0375, 0.075)), addUndoState=False)
s.Line(point1=(-0.325, 0.075), point2=(-0.325, 0.225000000046566))
s.VerticalConstraint(entity=g.findAt((-0.325, 0.15)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((0.0375, 0.075)), entity2=g.findAt(
    (-0.325, 0.15)), addUndoState=False)
s.CoincidentConstraint(entity1=v.findAt((-0.325, 0.075)), entity2=g.findAt((
    0.0375, 0.075)), addUndoState=False)
s.Line(point1=(0.15, 0.075), point2=(0.15, 0.25))
s.VerticalConstraint(entity=g.findAt((0.15, 0.1625)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((0.0375, 0.075)), entity2=g.findAt(
    (0.15, 0.1625)), addUndoState=False)
s.CoincidentConstraint(entity1=v.findAt((0.15, 0.075)), entity2=g.findAt((
    0.0375, 0.075)), addUndoState=False)
```

```
s.Line(point1=(0.55, 0.075), point2=(0.55, 0.25))
s.VerticalConstraint(entity=g.findAt((0.55, 0.1625)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((0.0375, 0.075)), entity2=g.findAt(
    (0.55, 0.1625)), addUndoState=False)
s.CoincidentConstraint(entity1=v.findAt((0.55, 0.075)), entity2=g.findAt((
    0.0375, 0.075)), addUndoState=False)
s.EqualLengthConstraint(entity1=g.findAt((-0.325, 0.15)), entity2=g.findAt((
    0.15, 0.1625)))
s.EqualLengthConstraint(entity1=g.findAt((0.15, 0.15625)), entity2=g.findAt((
    0.55, 0.1625)), addUndoState=False)
s.ObliqueDimension(
    vertex1=v.findAt((-0.325, 0.075)), vertex2=v.findAt((-0.325,
    0.225)), textPoint=(-0.829278707504272, 0.192073285579681), value=0.1)
s.move(vector=(0.0, 0.025), objectList=(g.findAt((0.0375, 0.075)), g.findAt((
    -0.325, 0.125)), g.findAt((0.15, 0.125)), g.findAt((0.55, 0.125))))
s.breakCurve(curve1=g.findAt((0.0375, 0.1)), point1=(-0.441734611988068,
    0.0968574285507202), curve2=g.findAt((-0.325, 0.15)), point2=(
    -0.326785087585449, 0.149390280246735))
s.breakCurve(curve1=g.findAt((0.2, 0.1)), point1=(0.00492632389068604,
    0.106707394123077), curve2=g.findAt((0.15, 0.15)), point2=(
    0.152718663215637, 0.155956923961639))
s.breakCurve(curve1=g.findAt((0.4375, 0.1)), point1=(0.415460348129272,
    0.100140750408173), curve2=g.findAt((0.55, 0.15)), point2=(
    0.556684136390686, 0.139540374279022))
s.HorizontalDimension(vertex1=v.findAt((-0.65, 0.1)), vertex2=v.findAt((0.725,

    0.1)), textPoint=(0.458155989646912, -0.165806710720062), value=2.0)
s.FixedConstraint(entity=g.findAt((0.15, 0.15)))
s.EqualLengthConstraint(entity1=g.findAt((-0.8, 0.1)), entity2=g.findAt((
    -0.0875, 0.1)))
s.delete(objectList=(c[34], ))
s.EqualLengthConstraint(entity1=g.findAt((-0.0875, 0.1)), entity2=g.findAt((
    0.35, 0.1)), addUndoState=False)
s.EqualLengthConstraint(entity1=g.findAt((0.3125, 0.1)), entity2=g.findAt((
    0.9125, 0.1)), addUndoState=False)
s.move(vector=(0.0375, 0.0), objectList=(g.findAt((-0.325, 0.15)), g.findAt((
    0.175, 0.15)), g.findAt((0.675, 0.15)), g.findAt((-0.575, 0.1)), g.findAt((

    -0.075, 0.1)), g.findAt((0.425, 0.1)), g.findAt((0.925, 0.1))))
s.move(
    vector=(-0.2125, -0.1),
    objectList=(
           g.findAt((-0.2875, 0.15)), g.findAt((0.2125, 0.15)),
           g.findAt((0.7125, 0.15)), g.findAt((-0.5375, 0.1)),
           g.findAt((-0.0375, 0.1)), g.findAt((0.4625, 0.1)),
           g.findAt((0.9625, 0.1))))
p = mdb.models['Model-1'].Part(name='Plate', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Plate']
p.BaseShellExtrude(sketch=s, depth=2.0)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Plate']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']


session.viewports['Viewport: 1'].setValues(displayedObject=p)
session.viewports['Viewport: 1'].partDisplay.setValues(renderStyle=SHADED)

mdb.models['Model-1'].Material('Steel')
mdb.models['Model-1'].materials['Steel'].Density(table=((7800.0, ), ))
```

```
mdb.models['Model-1'].materials['Steel'].Elastic(table=((210.0E9, 0.3), ))
mdb.models['Model-1'].materials['Steel'].Plastic(table=((300.0E6, 0.0),
    (350.0E6, 0.025), (375.0E6, 0.1), (394.0E6, 0.2), (400.0E6, 0.35)))

mdb.models['Model-1'].HomogeneousShellSection(name='PlateSection',
    preIntegrate=OFF, material='Steel', thickness=0.025,
    poissonDefinition=DEFAULT, temperature=GRADIENT,
    integrationRule=SIMPSON, numIntPts=5)
mdb.models['Model-1'].HomogeneousShellSection(name='StiffSection',
    preIntegrate=OFF, material='Steel', thickness=0.0125,
    poissonDefinition=DEFAULT, temperature=GRADIENT,
    integrationRule=SIMPSON, numIntPts=5)

f = p.faces
faces = f.findAt(
    ((0.833333, 0.0, 1.333333), ),
    ((0.333333, 0.0, 1.333333), ),
    ((-0.166667, 0.0, 1.333333), ),
    ((-0.666667, 0.0, 1.333333), ))
region =(None, None, faces, None)
p.SectionAssignment(region=region, sectionName='PlateSection', offset=-0.5)

faces = f.findAt(
    ((-0.5, 0.066669, 1.333333), ),
    ((0.0, 0.066671, 1.333333), ),
    ((0.5, 0.066673, 1.333333), ))
region =(None, None, faces, None)

p.SectionAssignment(region=region, sectionName='StiffSection')

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

a.DatumCsysByDefault(CARTESIAN)
a.Instance(name='Plate-1', part=p, dependent=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(renderStyle=SHADED)

p = mdb.models['Model-1'].parts['Plate']
e = p.edges
pickedEdges = e.findAt(((0.0, 0.0, 0.5), ))
p.PartitionEdgeByParam(edges=pickedEdges, parameter=0.5)

v = a.instances['Plate-1'].vertices
verts = v.findAt(((0.0, 0.0, 1.0), ))
a.Set(vertices=verts, name='Center')

e = a.instances['Plate-1'].edges
edges = e.findAt(
    ((0.875, 0.0, 2.0), ), ((0.625, 0.0, 0.0), ),
    ((1.0, 0.0, 0.5), ), ((0.375, 0.0, 2.0), ),
    ((0.125, 0.0, 0.0), ), ((-0.125, 0.0, 2.0), ),
    ((-0.375, 0.0, 0.0), ), ((-0.625, 0.0, 2.0), ),
    ((-1.0, 0.0, 0.5), ), ((-0.875, 0.0, 0.0), ))
a.Set(edges=edges, name='Edge')

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
    predefinedFields=OFF)

mdb.models['Model-1'].ExplicitDynamicsStep(name='Blast', previous='Initial',
    description='Apply blast loading', timePeriod=0.05, adiabatic=OFF,
    timeIncrementationMethod=AUTOMATIC_GLOBAL, scaleFactor=1,
```

```
    maxIncrement=None, massScaling=PREVIOUS_STEP, linearBulkViscosity=0.06,
    quadBulkViscosity=1.2)

regionDef=a.sets['Center']
mdb.models['Model-1'].steps['Blast'].Monitor(dof=2, node=regionDef)

mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    numIntervals=25)

mdb.models['Model-1'].historyOutputRequests['H-Output-1'].setValues(
    numIntervals=500)

regionDef=mdb.models['Model-1'].rootAssembly.sets['Center']
mdb.models['Model-1'].HistoryOutputRequest(name='Center-U2',
    createStepName='Blast', variables=('U2', ), numIntervals=500,
    region=regionDef)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)

region = a.sets['Edge']
mdb.models['Model-1'].EncastreBC(name='Fix edges', createStepName='Initial',
    region=region)
mdb.models['Model-1'].TabularAmplitude(name='Blast', timeSpan=STEP,
    smooth=0.25, data=((0.0, 0.0), (0.001, 7.0E5), (0.01, 7.0E5),
    (0.02, 0.0), (0.05, 0.0)))

s = a.instances['Plate-1'].faces
side1Faces = s.findAt(
    ((0.833333, 0.0, 1.333333), ),
    ((0.333333, 0.0, 1.333333), ),
    ((-0.166667, 0.0, 1.333333), ),
    ((-0.666667, 0.0, 1.333333), ))
region = regionToolset.Region(side1Faces=side1Faces)
mdb.models['Model-1'].Pressure(name='Pressure load', createStepName='Blast',
    region=region, magnitude=1.0, amplitude='Blast')

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
p.seedPart(size=0.1)

e = p.edges
edges =(e.findAt(coordinates=(-0.5, 0.025, 2.0)), e.findAt(coordinates=(
    0.0, 0.025, 2.0)), e.findAt(coordinates=(0.5, 0.025, 2.0)), e.findAt(
    coordinates=(-0.5, 0.075, 0.0)), e.findAt(coordinates=(0.0, 0.075,
    0.0)), e.findAt(coordinates=(0.5, 0.075, 0.0)))
p.seedEdgeByNumber(edges=edges, number=2)

elemType1 = mesh.ElemType(elemCode=S4R, elemLibrary=EXPLICIT,
    hourglassControl=RELAX_STIFFNESS)
elemType2 = mesh.ElemType(elemCode=S3, elemLibrary=EXPLICIT)

f = p.faces
faces = f

regions =(None, None, faces, None)
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2))

pickedRegions = f
p.setMeshControls(regions=pickedRegions, elemShape=QUAD, algorithm=MEDIAL_AXIS)
```

```
p.generateMesh()

session.viewports['Viewport: 1'].assemblyDisplay.setValues(renderStyle=SHADED)

mdb.Job(name='BlastLoad', model='Model-1')
mdb.jobs['BlastLoad'].setValues(description='Blast load on a flat plate with
stiffeners: S4R elements (20x20 mesh) Normal stiffeners (20x2)')

a.regenerate()

mdb.saveAs('StiffPlate')
```

Axisymmetric mount

This Python script can be used to create a model for examples contained in this guide.

- gsi_mount_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_mount_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for rubber mount example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

##
##  Sketch profile of the mount
##
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=0.3)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3, viewStyle=AXISYM)
s.setPrimaryObject(option=STANDALONE)
s.ConstructionLine(point1=(0.0, -0.15), point2=(0.0, 0.15))
s.FixedConstraint(entity=g.findAt((0.0, 0.0)))
s.rectangle(point1=(0.01375, 0.035), point2=(0.04625, 0.0175))
s.DistanceDimension(entity1=g.findAt((0.0, 0.0)), entity2=v.findAt((0.01375,
    0.0175)), textPoint=(0.012268845923245, 0.0116447377949953), value=0.01)
s.ObliqueDimension(vertex1=v.findAt((0.01, 0.035)), vertex2=v.findAt((0.01,
    0.0175)), textPoint=(0.00531696528196335, 0.0242882780730724), value=0.03)
s.ObliqueDimension(vertex1=v.findAt((0.01, 0.0175)), vertex2=v.findAt((0.0425,

    0.0175)), textPoint=(0.0255863033235073, 0.0116447377949953), value=0.05)
```

```
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].view.setValues(nearPlane=0.128767,
    farPlane=0.166316, width=0.111144, height=0.0935352)
s.CircleByCenterPerimeter(center=(0.07375, 0.04875), point1=(0.06, 0.0275))
s.CoincidentConstraint(entity1=v.findAt((0.06, 0.0275)), entity2=g.findAt((
    0.06, 0.0325)), addUndoState=False)
s.DistanceDimension(entity1=g.findAt((0.0, 0.0)), entity2=v.findAt((0.07375,
   0.04875)), textPoint=(0.0690968036651611, -0.00352155975997448), value=0.1)
session.viewports['Viewport: 1'].view.fitView()
s.VerticalDimension(vertex1=v.findAt((0.06, 0.0475)), vertex2=v.findAt((0.1,
    0.04875)), textPoint=(0.117072999477386, 0.0480493120849133), value=0.0)
s.VerticalDimension(vertex1=v.findAt((0.06, 0.0275)), vertex2=v.findAt((0.06,
    0.0175)), textPoint=(0.0739211142063141, 0.0188609156757593), value=0.005)
s.autoTrimCurve(curve1=g.findAt((0.14, 0.0725)), point1=(0.0657630488276482,
    0.0819593593478203))
s.autoTrimCurve(curve1=g.findAt((0.06, 0.0325)), point1=(0.0606105849146843,
    0.0390352495014668))
s.autoTrimCurve(curve1=g.findAt((0.035, 0.0475)), point1=(0.0567462332546711,
    0.0471908301115036))
s.move(vector=(0.0, -0.0175), objectList=(g.findAt((0.01, 0.0325)), g.findAt((
    0.035, 0.0175)), g.findAt((0.054658, 0.034496)), g.findAt((0.06, 0.02)),
    g.findAt((0.031415, 0.0475))))
p = mdb.models['Model-1'].Part(name='Mount', dimensionality=AXISYMMETRIC,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Mount']
p.BaseShell(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Mount']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

##
##  Create material 'Rubber'
##
mdb.models['Model-1'].Material('Rubber')
mdb.models['Model-1'].materials['Rubber'].Hyperelastic(type=POLYNOMIAL,
    table=())
mdb.models['Model-1'].materials['Rubber'].hyperelastic.UniaxialTestData(table=((

    0.054E6, 0.0380), (0.152E6, 0.1338), (0.254E6, 0.2210), (0.362E6, 0.3450),

    (0.459E6, 0.4600), (0.583E6, 0.6242), (0.656E6, 0.8510), (0.730E6,
    1.4268)))
mdb.models['Model-1'].materials['Rubber'].hyperelastic.BiaxialTestData(table=((

    0.089E6, 0.0200), (0.255E6, 0.1400), (0.503E6, 0.4200), (0.958E6, 1.4900),

    (1.703E6, 2.7500), (2.413E6, 3.4500)))
mdb.models['Model-1'].materials['Rubber'].hyperelastic.PlanarTestData(table=((
    0.055E6, 0.0690), (0.324E6, 0.2828), (0.758E6, 1.3862), (1.269E6, 3.0345),

    (1.779E6, 4.0621)))
##
##  Create material 'Steel'
##
mdb.models['Model-1'].Material('Steel')
mdb.models['Model-1'].materials['Steel'].Elastic(table=((200.E9, 0.3), ))
##
##  Create solid sections for the rubber and steel
##
mdb.models['Model-1'].HomogeneousSolidSection(name='RubberSection',
```

```
    material='Rubber', thickness=1.0)
mdb.models['Model-1'].HomogeneousSolidSection(name='SteelSection',
    material='Steel', thickness=1.0)
##
##  Partition the part into two regions (rubber and steel regions)
##
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

t = p.MakeSketchTransform(
    sketchPlane=f.findAt(coordinates=(0.043333, 0.001667, 0.0),
    normal=(0.0, 0.0, 1.0)), sketchPlaneSide=SIDE1,
    origin=(0.033052, 0.014514, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=0.134, gridSpacing=0.003, transform=t)
g, v, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)

s.Line(point1=(0.026948, -0.009514), point2=(-0.03, -0.009514))
s.HorizontalConstraint(entity=g.findAt((-0.001526, -0.009514)))
s.PerpendicularConstraint(entity1=g.findAt((0.026948, -0.012014)),
    entity2=g.findAt((-0.001526, -0.009514)))

pickedFaces = f.findAt(((0.043333, 0.001667, 0.0), ))
p.PartitionFaceBySketch(faces=pickedFaces, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

##
##  Assign rubber section
##
p = mdb.models['Model-1'].parts['Mount']
f = p.faces

faces = f.findAt(((0.042303, 0.006937, 0.0), ))
region = regionToolset.Region(faces=faces)
p.SectionAssignment(region=region, sectionName='RubberSection', offset=0.0)
##
##  Assign steel section
##
faces = f.findAt(((0.043333, 0.003333, 0.0), ))
region = regionToolset.Region(faces=faces)
p.SectionAssignment(region=region, sectionName='SteelSection', offset=0.0)

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the mount
##
p = mdb.models['Model-1'].parts['Mount']
a.Instance(name='Mount-1', part=p, dependent=ON)
##
##  Create geometry set 'Middle'
##
e = a.instances['Mount-1'].edges
```

```
edges = e.findAt(((0.020708, 0.03, 0.0), ))
a.Set(edges=edges, name='Middle')
##
##  Create geometry set 'Out'
##
v = a.instances['Mount-1'].vertices
verts = v.findAt(((0.01, 0.0, 0.0), ))
a.Set(vertices=verts, name='Out')
##
##  Create surface 'Bottom'
##
s = a.instances['Mount-1'].edges
side1Edges = s.findAt(((0.0475, 0.0, 0.0), ))
a.Surface(side1Edges=side1Edges, name='Bottom')

##
##  Create a static general step
##
mdb.models['Model-1'].StaticStep(name='Compress mount', previous='Initial',
    description='Apply axial pressure load to mount', timePeriod=1,
    adiabatic=OFF, maxNumInc=100, stabilization=None,
    timeIncrementationMethod=AUTOMATIC,
    initialInc=0.01, minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT,
    amplitude=RAMP, extrapolation=LINEAR, fullyPlastic="", nlgeom=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    step='Compress mount')
##
##  Modify output requests
##
mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=('S', 'PE', 'PEEQ', 'PEMAG', 'NE', 'LE', 'U', 'RF',
    'CF', 'CSTRESS', 'CDISP'))

regionDef=a.sets['Out']
mdb.models['Model-1'].HistoryOutputRequest(name='H-Output-1',
    createStepName='Compress mount', variables=('U1', 'U2', 'U3'),
    region=regionDef)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
##
##  Apply pressure load
##
region = a.surfaces['Bottom']
mdb.models['Model-1'].Pressure(name='Pressure',
    createStepName='Compress mount', region=region, magnitude=500000.0)
##
##  Apply symmetry bc to set "Middle'
##
region = a.sets['Middle']
mdb.models['Model-1'].DisplacementBC(name='Symmetry',
    createStepName='Compress mount', region=region, u2=0.0)
##
##  Suppress visibility of datum geometry
##
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
   geometryEdgesInShaded=OFF, datumPoints=OFF, datumAxes=OFF, datumPlanes=OFF,

    datumCoordSystems=OFF)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON, loads=OFF,
```

```
    bcs=OFF, predefinedFields=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
##
##  Assign edge seeds
##
p = mdb.models['Model-1'].parts['Mount']
e = p.edges

pickedEdges = e.findAt(((0.0225, 0.005, 0.0), ), ((0.0475, 0.0, 0.0), ),
    ((0.020708, 0.03, 0.0), ))
p.seedEdgeByNumber(edges=pickedEdges, number=30)

pickedEdges = e.findAt(((0.053289, 0.023434, 0.0), ), ((0.01, 0.01125, 0.0),
))
p.seedEdgeByNumber(edges=pickedEdges, number=14)

pickedEdges = e.findAt(((0.01, 0.00125, 0.0), ), ((0.06, 0.00375, 0.0), ))
p.seedEdgeByNumber(edges=pickedEdges, number=1)
##
##  Use structured meshing
##
f = p.faces
pickedRegions = f
p.setMeshControls(regions=pickedRegions, technique=STRUCTURED)
##
##  Assign element type to the rubber
##
elemType1 = mesh.ElemType(elemCode=CAX4H, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=CAX3, elemLibrary=STANDARD)
faces = f.findAt(((0.042303, 0.006937, 0.0), ))
pickedRegions =(faces, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
##
##  Assign element type to the steel
##
elemType1 = mesh.ElemType(elemCode=CAX4I, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=CAX3, elemLibrary=STANDARD)
faces = f.findAt(((0.043333, 0.003333, 0.0), ))
pickedRegions =(faces, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
##
##  Generate mesh
##
p.generateMesh()

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
##
##  Create job
##
mdb.Job(name='Mount', model='Model-1',
    description='Axisymmetric mount analysis under axial loading',
    modelPrint=ON)

a = mdb.models['Model-1'].rootAssembly
a.regenerate()
##
##  Save model database
##
```

```
mdb.saveAs('Mount')
```

Vibration of a piping system

This Python script can be used to create a model for examples contained in this guide.

- gsi_pipe_caemodel.py

Fetching the Python script

The script is included in the Abaqus release; you can extract it from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File->Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_pipe_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for pipe example
#
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()
##
##  Rename the model 'Original'
##
session.viewports['Viewport: 1'].setValues(displayedObject=None)
mdb.models.changeKey('Model-1', 'Original')

##
##  Sketch profile of the pipe
##
s = mdb.models['Original'].ConstrainedSketch(name='__profile__',
    sheetSize=20.0)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=20.0, gridSpacing=0.5, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.5,
    decimalPlaces=2)
s.Line(point1=(0.0, 0.0), point2=(5.0, 0.0))
p = mdb.models['Original'].Part(name='Pipe', dimensionality=THREE_D,
```

```
    type=DEFORMABLE_BODY)
p = mdb.models['Original'].parts['Pipe']
p.BaseWire(sketch=s)
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Original'].sketches['__profile__']

##
##  Create material 'Steel'
##
mdb.models['Original'].Material('Steel')
mdb.models['Original'].materials['Steel'].Elastic(table=((2.E+11, 0.3), ))
mdb.models['Original'].materials['Steel'].Density(table=((7800.0, ), ))
##
##  Create pipe profile and section
##
mdb.models['Original'].PipeProfile(name='PipeProfile', r=0.09, t=0.02)
mdb.models['Original'].BeamSection(name='PipeSection', profile='PipeProfile',
    integration=DURING_ANALYSIS, material='Steel', temperatureVar=LINEAR)
##
##  Assign pipe section
##
e = p.edges
region=(None, e.findAt(((1.25, 0.0, 0.0), ), ), None, None)
p.SectionAssignment(region=region, sectionName='PipeSection')
##
##  Assign beam orientations
##
region=(None, e.findAt(((1.25, 0.0, 0.0), ), ), None, None)
p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(0.0, 0.0, -1.0))

a = mdb.models['Original'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the pipe
##
p = mdb.models['Original'].parts['Pipe']
a.Instance(name='Pipe-1', part=p, dependent=ON)
##
##  Create geometry set 'Left'
##
v = a.instances['Pipe-1'].vertices
a.Set(vertices=v.findAt(((0.0, 0.0, 0.0), ), ), name='Left')
##
##  Create geometry set 'Right'
##
a.Set(vertices=v.findAt(((5.0, 0.0, 0.0), ), ), name='Right')
##
##  Create a static general step
##
mdb.models['Original'].StaticStep(name='Pull I', previous='Initial',
    description='Apply axial tensile load of 4.0 MN', timePeriod=1,
    adiabatic=OFF, maxNumInc=100, stabilization=None,
    timeIncrementationMethod=AUTOMATIC,
    initialInc=0.1, minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT,
    amplitude=RAMP, extrapolation=LINEAR, fullyPlastic="", nlgeom=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Pull I')
```

```
##
##  Create a frequency extraction step
##
mdb.models['Original'].FrequencyStep(name='Frequency I', previous='Pull I',
    description='Extract modes and frequencies', numEigen=8,
    eigensolver=LANCZOS)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Pull I')
##
##  Modify output requests
##
mdb.models['Original'].fieldOutputRequests['F-Output-1'].setValues(
    variables=PRESELECT)
del mdb.models['Original'].historyOutputRequests['H-Output-1']
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Frequency I')
mdb.models['Original'].fieldOutputRequests['F-Output-2'].setValues(
    variables=PRESELECT)
mdb.models['Original'].steps['Pull I'].Restart(frequency=10, overlay=OFF)
mdb.models['Original'].steps['Frequency I'].Restart(frequency=1)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Pull I')
##
##  Apply encastre bc to left end
##
region = a.sets['Left']
mdb.models['Original'].EncastreBC(name='Left end', createStepName='Pull I',
    region=region)
##
##  Fix all dofs at right end except for U1
##
region = a.sets['Right']
mdb.models['Original'].DisplacementBC(name='Right end',
    createStepName='Pull I',
    region=region, u2=0.0, u3=0.0, ur1=0.0, ur2=0.0, ur3=0.0)
##
##  Apply concentrated force to right end
##
region = a.sets['Right']
mdb.models['Original'].ConcentratedForce(name='Force',
    createStepName='Pull I', region=region, cf1=4.e6)

##
##  Assign edge seed
##
p = mdb.models['Original'].parts['Pipe']
e = p.edges
edges=(e.findAt((1.25, 0.0, 0.0), ), )
p.seedEdgeByNumber(edges=edges, number=30)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=PIPE32)
regions=(None,
    e.findAt(((1.25, 0.0, 0.0), ), ), None, None)
p.setElementType(regions=regions, elemTypes=(elemType1, ))
##
##  Generate mesh
##
p.generateMesh()

##
```

```
##  Create job
##
mdb.Job(name='Pipe', model='Original',
    description='Analysis of a 5 meter long pipe under axial load')
##
##  Save model database
##
mdb.saveAs('Pipe')
##
##  Copy model to 'Restart'
##
mdb.Model('Restart', mdb.models['Original'])
##
##  Edit restart model attributes
##
mdb.models['Restart'].setValues(restartJob='Pipe', restartStep='Frequency I',
    restartIncrement=STEP_END)

a = mdb.models['Restart'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Create general static step; insert after previous frequency step
##
mdb.models['Restart'].StaticStep(name='Pull II', previous='Frequency I',
    description='Apply axial tensile load of 8.0 MN', timePeriod=1,
    adiabatic=OFF, maxNumInc=100,
    stabilizationMethod=NONE, timeIncrementationMethod=AUTOMATIC,
    initialInc=0.1, minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT,
    amplitude=RAMP, extrapolation=LINEAR, fullyPlastic="")
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Pull II')
##
##  Create frequency extraction step
##
mdb.models['Restart'].FrequencyStep(name='Frequency II', previous='Pull II',
    description='Extract modes and frequencies', numEigen=8,
    eigensolver=LANCZOS)
##
##  Modify output requests
##
mdb.models['Restart'].steps['Pull II'].Restart(frequency=10, overlay=OFF)
mdb.models['Restart'].steps['Frequency II'].Restart(frequency=1)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Frequency II')

session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
    predefinedFields=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Pull II')
##
##  Double the magnitude of the applied force
##  in the second general step
##
mdb.models['Restart'].loads['Force'].setValuesInStep(stepName='Pull II',
    cf1=8.e6)
##
##  Create a restart analysis job
##
mdb.Job(name='PipeRestart', model='Restart', type=RESTART,
    explicitPrecision=SINGLE,
    description='Restart analysis of a 5 meter long pipe under axial load',
    userSubroutine='', numCpus=1, scratch='', echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
#
```

```
a = mdb.models['Original'].rootAssembly
a.regenerate()

a = mdb.models['Restart'].rootAssembly
a.regenerate()
##
##  Save model database
##
mdb.saveAs('Pipe')
```

Forming a channel

These Python scripts can be used to create models for examples contained in this guide.

- gsi_channel_caemodel.py
- gsi_channel_caemodel_spring.py

Fetching the Python script

The scripts are included in the Abaqus release; you can extract them from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File**->**Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_channel_caemodel.py

```
#
# Getting Started with Abaqus
#
# Script for forming portion channel example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

mdb.models.changeKey(fromName='Model-1', toName='standard')

##
##  Sketch profile of blank
##
s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.25)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=0.25, gridSpacing=0.005, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.005,
    decimalPlaces=3)
s.Line(point1=(-0.05, -0.005), point2=(0.05, -0.005))
s.Line(point1=(0.05, -0.005), point2=(0.05, 0.005))
s.Line(point1=(0.05, 0.005), point2=(-0.05, 0.005))
s.Line(point1=(-0.05, 0.005), point2=(-0.05, -0.005))
```

```
s.HorizontalDimension(vertex1=v.findAt((-0.05, 0.005)), vertex2=v.findAt((
    0.05, 0.005)), textPoint=(0.0457058809697628, 0.0127279916778207),
    value=0.1)
s.FixedConstraint(entity=g.findAt((0.0, -0.005)))
s.PerpendicularConstraint(entity1=g.findAt((-0.05, 0.0)), entity2=g.findAt((
    0.0, 0.005)))
s.PerpendicularConstraint(entity1=g.findAt((0.05, 0.0)), entity2=g.findAt((
    0.0, 0.005)))
s.VerticalDimension(vertex1=v.findAt((-0.05, -0.005)), vertex2=v.findAt((
    -0.05, 0.005)), textPoint=(-0.0521911755204201, 0.000539974775165319),
    value=0.001)
p = mdb.models['standard'].Part(name='Blank', dimensionality=TWO_D_PLANAR,
    type=DEFORMABLE_BODY)
p = mdb.models['standard'].parts['Blank']
p.BaseShell(sketch=s)
p = mdb.models['standard'].parts['Blank']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']
##
##  Sketch profile of punch
##
s0 = mdb.models['standard'].ConstrainedSketch(name='__profile__', sheetSize=0.25)
g, v, d = s0.geometry, s0.vertices, s0.dimensions
s0.sketchOptions.setValues(sheetSize=0.25, gridSpacing=0.005, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.005,
    decimalPlaces=3)
s0.Line(point1=(-0.06, 0.0), point2=(0.0, 0.0))
s0.Line(point1=(0.0, 0.0), point2=(0.0, 0.06))
s0.FilletByRadius(radius=0.005, curve1=g.findAt((-0.03, 0.0)), nearPoint1=(
    -0.00207383744418621, 0.000262239947915077), curve2=g.findAt((0.0, 0.03)),
    nearPoint2=(-0.00016310065984726, 0.00246503762900829))
session.viewports['Viewport: 1'].view.fitView()
mdb.models['standard'].ConstrainedSketch(name='Punch', objectToCopy=s0)
p = mdb.models['standard'].Part(name='Punch', dimensionality=TWO_D_PLANAR,
    type=ANALYTIC_RIGID_SURFACE)
p = mdb.models['standard'].parts['Punch']
p.AnalyticRigidSurf2DPlanar(sketch=s0)
p = mdb.models['standard'].parts['Punch']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']
#
#   Create holder
#
s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.25)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=STANDALONE)
s.sketchOptions.setValues(gridOrigin=(-0.026910743678459, 0.026910743678459))
s.retrieveSketch(sketch=mdb.models['standard'].sketches['Punch'])
s.mirror(mirrorLine=g.findAt((0.0, 0.0325)), objectList=(g.findAt((-0.0325,
    0.0)), g.findAt((0.0, 0.0325)), g.findAt((-0.001464, 0.001464))))
p = mdb.models['standard'].Part(name='Holder', dimensionality=TWO_D_PLANAR,
    type=ANALYTIC_RIGID_SURFACE)
p = mdb.models['standard'].parts['Holder']
p.AnalyticRigidSurf2DPlanar(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['standard'].parts['Holder']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']
```

```
#
#   Create die
#
s = mdb.models['standard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.25)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=STANDALONE)
s.sketchOptions.setValues(gridOrigin=(-0.026910743678459, 0.026910743678459))

s.retrieveSketch(sketch=mdb.models['standard'].sketches['Punch'])

s.mirror(mirrorLine=g.findAt((0.0, 0.0325)), objectList=(g.findAt((-0.0325,
    0.0)), g.findAt((0.0, 0.0325)), g.findAt((-0.001464, 0.001464))))

s.mirror(mirrorLine=g.findAt((0.0325, 0.0)), objectList=(g.findAt((0.0325,
    0.0)), g.findAt((0.0, 0.0325)), g.findAt((0.001464, 0.001464))))

p = mdb.models['standard'].Part(name='Die', dimensionality=TWO_D_PLANAR,
    type=ANALYTIC_RIGID_SURFACE)
p = mdb.models['standard'].parts['Die']
p.AnalyticRigidSurf2DPlanar(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['standard'].parts['Die']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['standard'].sketches['__profile__']

##
##  Add reference point to die
##
p = mdb.models['standard'].parts['Die']
e = p.edges
p.ReferencePoint(point=p.InterestingPoint(
    edge=e.findAt(coordinates=(0.003087, -0.000381, 0.0)), rule=CENTER))
##
##  Add reference point to holder
##
p = mdb.models['standard'].parts['Holder']
e = p.edges
p.ReferencePoint(point=p.InterestingPoint(
    edge=e.findAt(coordinates=(0.003087, 0.000381, 0.0)), rule=CENTER))
##
##  Add reference point to punch
##
p = mdb.models['standard'].parts['Punch']
e = p.edges
p.ReferencePoint(point=p.InterestingPoint(
    edge=e.findAt(coordinates=(-0.003087, 0.000381, 0.0)), rule=CENTER))
##
##  Create material 'Steel'
##
mdb.models['standard'].Material('Steel')
mdb.models['standard'].materials['Steel'].Elastic(table=((2.1E11, 0.3), ))
mdb.models['standard'].materials['Steel'].Plastic(table=((400.E6, 0.0), (
    420.E6, 0.02), (500.E6, 0.2), (600.E6, 0.5)))
##
##  Create and assign solid section to blank
##
mdb.models['standard'].HomogeneousSolidSection(name='BlankSection',
    material='Steel', thickness=1.0)
```

```
p = mdb.models['standard'].parts['Blank']
f = p.faces
faces = f.findAt(((-0.016667, -0.004667, 0.0), ))
region = regionToolset.Region(faces=faces)
p = mdb.models['standard'].parts['Blank']
p.SectionAssignment(region=region, sectionName='BlankSection', offset=0.0)
##
##  Define datum coordinate system
##
p = mdb.models['standard'].parts['Blank']
p.DatumCsysByDefault(coordSysType=CARTESIAN)
##
##  Assign material orientation to blank
##
p = mdb.models['standard'].parts['Blank']
f = p.faces
faces = f.findAt(((-0.016667, -0.004667, 0.0), ))
region = regionToolset.Region(faces=faces)
datum = p.datums[3]
p.MaterialOrientation(region=region, localCsys=datum)

a = mdb.models['standard'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
##
##  Set coordinate system (done by default)
##
a = mdb.models['standard'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
##
##  Instance the blank
##
p = mdb.models['standard'].parts['Blank']
a.Instance(name='Blank-1', part=p, dependent=ON)
a = mdb.models['standard'].rootAssembly
##
##  Instance the punch
##
p = mdb.models['standard'].parts['Punch']
a.Instance(name='Punch-1', part=p, dependent=ON)
##
##  Translate the punch
##
a = mdb.models['standard'].rootAssembly
p = a.instances['Punch-1']
p.translate(vector=(0.116, 0.0, 0.0))
##
##   Edge to edge constraints between punch and blank
##
e11 = a.instances['Punch-1'].edges
e12 = a.instances['Blank-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.06975, 0.0, 0.0)),
    fixedAxis=e12.findAt(coordinates=(0.025, -0.004, 0.0)), flip=ON,
    clearance=0.0)
session.viewports['Viewport: 1'].view.fitView()
e11 = a.instances['Punch-1'].edges
e12 = a.instances['Blank-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.116, 0.01575, 0.0)),
    fixedAxis=e12.findAt(coordinates=(-0.05, -0.00425, 0.0)), flip=ON,
    clearance=-0.05)
##
##  Instance the holder
```

```
##
p = mdb.models['standard'].parts['Holder']
a.Instance(name='Holder-1', part=p, dependent=ON)
p = a.instances['Holder-1']
p.translate(vector=(0.056, 0.0, 0.0))
session.viewports['Viewport: 1'].view.fitView()
##
##   Edge to edge constraints between holder and punch
##
e11 = a.instances['Holder-1'].edges
e12 = a.instances['Punch-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.056, 0.01875, 0.0)),
    fixedAxis=e12.findAt(coordinates=(0.0, 0.01575, 0.0)), flip=OFF,
    clearance=0.001)

e11 = a.instances['Holder-1'].edges
e12 = a.instances['Blank-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.04725, 0.0, 0.0)),
    fixedAxis=e12.findAt(coordinates=(0.025, -0.004, 0.0)), flip=OFF,
    clearance=0.0)
##
##  Instance the die
##
p = mdb.models['standard'].parts['Die']
a.Instance(name='Die-1', part=p, dependent=ON)
p = a.instances['Die-1']
p.translate(vector=(0.067, 0.0, 0.0))
session.viewports['Viewport: 1'].view.fitView()
##
##   Edge to edge constraints between die and blank (horizontal)
##
e11 = a.instances['Die-1'].edges
e12 = a.instances['Blank-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.067, -0.01875, 0.0)),
    fixedAxis=e12.findAt(coordinates=(-0.05, -0.00425, 0.0)), flip=OFF,
    clearance=-0.051)

e11 = a.instances['Die-1'].edges
e12 = a.instances['Blank-1'].edges
a.EdgeToEdge(movableAxis=e11.findAt(coordinates=(0.04725, 0.0, 0.0)),
    fixedAxis=e12.findAt(coordinates=(-0.025, -0.005, 0.0)), flip=ON,
    clearance=0.0)

p = a.instances['Blank-1']
p.translate(vector=(0.0, 0.009, 0.0))

a = mdb.models['standard'].rootAssembly
a.regenerate()

#
#   Create geometry set RefPunch
#
a = mdb.models['standard'].rootAssembly
r1 = a.instances['Punch-1'].referencePoints
refPoints1 = (r1[2], )
a.Set(referencePoints=refPoints1, name='RefPunch')
#
#   Create geometry set RefHolder
#
a = mdb.models['standard'].rootAssembly
r1 = a.instances['Holder-1'].referencePoints
```

```
refPoints1 = (r1[2], )
a.Set(referencePoints=refPoints1, name='RefHolder')
#
#   Create geometry set RefDie
#
a = mdb.models['standard'].rootAssembly
r1 = a.instances['Die-1'].referencePoints
refPoints1 = (r1[2], )
a.Set(referencePoints=refPoints1, name='RefDie')
#
#   Reset view
#
session.viewports['Viewport: 1'].view.setValues(width=0.0066561,
    height=0.0042551, cameraPosition=(-0.048954, 0.0037356, 0.34938),
    cameraTarget=(-0.048954, 0.0037356, 0))
#
#   Create geometry set Center
#
a = mdb.models['standard'].rootAssembly
e = a.instances['Blank-1'].edges
edges = e.findAt(((-0.05, 0.00475, 0.0), ))
a.Set(edges=edges, name='Center')
#
#   Create surface BlankTop
#
a = mdb.models['standard'].rootAssembly
e1 = a.instances['Blank-1'].edges
a.Surface(name='BlankTop', side1Edges=e1.findAt(((0.025, 0.005, 0.0), ), ))
#
#   Create surface BlankBot
#
a = mdb.models['standard'].rootAssembly
e1 = a.instances['Blank-1'].edges
a.Surface(name='BlankBot', side1Edges=e1.findAt(((-0.025, 0.004, 0.0), ), ))

session.viewports['Viewport: 1'].view.fitView()
#
#   Create surface PunchSurf
#
s = a.instances['Punch-1'].edges
side2Edges1 = s.findAt(((0.0, 0.02375, 0.0), ))
a.Surface(side2Edges=side2Edges1, name='PunchSurf')
#
#   Create surface HolderSurf
#
s = a.instances['Holder-1'].edges
side1Edges = s.findAt(((0.001, 0.02375, 0.0), ))
a.Surface(side1Edges=side1Edges, name='HolderSurf')

#
#   Create surface DieSurf
#
s = a.instances['Die-1'].edges
side2Edges = s.findAt(((0.001, -0.01475, 0.0), ))
a.Surface(side2Edges=side2Edges, name='DieSurf')

##
##  Create two static general steps
##
mdb.models['standard'].StaticStep(name='Holder force',
    previous='Initial',
```

```
    description='Apply holder force', timePeriod=1,
    adiabatic=OFF, maxNumInc=100, stabilization=None,
    timeIncrementationMethod=AUTOMATIC,
    initialInc=0.05, minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT,
    amplitude=RAMP, extrapolation=LINEAR, nlgeom=ON)
mdb.models['standard'].StaticStep(name='Move punch',
    previous='Holder force',
    description='Apply punch stroke', timePeriod=1,
    adiabatic=OFF, maxNumInc=1000, stabilization=None,
    timeIncrementationMethod=AUTOMATIC,
    initialInc=0.05, minInc=1e-05, maxInc=1, matrixSolver=SOLVER_DEFAULT,
    amplitude=RAMP, extrapolation=LINEAR)
##
##  Modify output requests
##
mdb.models['standard'].fieldOutputRequests['F-Output-1'].setValues(
    variables=PRESELECT, frequency=20)

regionDef=a.sets['RefPunch']
mdb.models['standard'].HistoryOutputRequest(name='H-Output-2',
    createStepName='Holder force',
    variables=('RF2', 'U2'),
    region=regionDef)
##
##  Create degree of freedom monitor
##
mdb.models['standard'].steps['Holder force'].Monitor(dof=2,
    node=regionDef, frequency=1)
mdb.models['standard'].steps['Move punch'].Monitor(dof=2,
    node=regionDef, frequency=1)
##
##  Print diagnostic data for contact
##
mdb.models['standard'].steps['Holder force'].DiagnosticPrint(frequency=1,
    contact=ON, plasticity=OFF, residual=ON, solve=OFF)
mdb.models['standard'].steps['Move punch'].DiagnosticPrint(frequency=1,
    contact=ON, plasticity=OFF, residual=ON, solve=OFF)
##
##  Reset view
##
session.viewports['Viewport: 1'].view.setValues(width=0.025757,
    height=0.016466, cameraPosition=(-0.04834, 0.0033255, 0.34366),
    cameraTarget=(-0.04834, 0.0033255, 0))

session.viewports['Viewport: 1'].assemblyDisplay.setValues(interactions=ON)
session.viewports['Viewport: 1'].view.fitView()
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Initial')
##
##  Create contact property 'Fric'
##
mdb.models['standard'].ContactProperty('Fric')
mdb.models['standard'].interactionProperties['Fric'].TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC, slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF, dependencies=0,
    table=((0.1, ), ), shearStressLimit=None, maximumElasticSlip=FRACTION,
    fraction=0.005, elasticSlipStiffness=None)
##
##  Create contact property 'NoFric'
##
mdb.models['standard'].ContactProperty('NoFric')
mdb.models['standard'].interactionProperties['NoFric'].TangentialBehavior(
```

```
    formulation=FRICTIONLESS)
##
##  Create contact interaction 'Punch-Blank'
##
region1=a.surfaces['PunchSurf']
region2=a.surfaces['BlankTop']
mdb.models['standard'].SurfaceToSurfaceContactStd(name='Punch-Blank',
    createStepName='Initial', main=region1, secondary=region2,
    sliding=FINITE, interactionProperty='NoFric')
##
##  Create contact interaction 'Holder-Blank'
##
region1=a.surfaces['HolderSurf']
region2=a.surfaces['BlankTop']
mdb.models['standard'].SurfaceToSurfaceContactStd(name='Holder-Blank',
    createStepName='Initial', main=region1, secondary=region2,
    sliding=FINITE, interactionProperty='Fric')
##
##  Create contact interaction 'Die-Blank'
##
region1=a.surfaces['DieSurf']
region2=a.surfaces['BlankBot']
mdb.models['standard'].SurfaceToSurfaceContactStd(name='Die-Blank',
    createStepName='Initial', main=region1, secondary=region2,
    sliding=FINITE, interactionProperty='Fric')

mdb.models['standard'].StdContactControl(name='stabilize',
    automaticTolerances=OFF,
    stabilizeChoice=AUTOMATIC,
    dampFactor=0.001)
mdb.models['standard'].interactions['Punch-Blank'].setValuesInStep(
    stepName='Move punch', contactControls='stabilize')
##
##
session.viewports['Viewport: 1'].view.fitView()
##
##  Create BCs in step "Establish contact"
##
region = a.sets['Center']
mdb.models['standard'].XsymmBC(name='CenterBC',
    createStepName='Holder force', region=region)
region = a.sets['RefDie']
mdb.models['standard'].DisplacementBC(name='RefDieBC',
    createStepName='Holder force', region=region,
    u1=0.0, u2=0.0, ur3=0.0)
region = a.sets['RefHolder']
mdb.models['standard'].DisplacementBC(name='RefHolderBC',
    createStepName='Holder force', region=region,
    u1=0.0, ur3=0.0)
region = a.sets['RefPunch']
mdb.models['standard'].DisplacementBC(name='RefPunchBC',
    createStepName='Holder force', region=region,
    u1=0.0, u2=0.0, ur3=0.0)
##
##  Add holder force
##
region = a.sets['RefHolder']
mdb.models['standard'].ConcentratedForce(name='RefHolderForce',
    createStepName='Holder force', region=region, cf2=-440000.0)
##
##  Push punch down in step  "Move punch"
```

```
##
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Move punch')
mdb.models['standard'].boundaryConditions['RefPunchBC'].setValuesInStep(
    stepName='Move punch', u2=-0.03)

session.viewports['Viewport: 1'].view.fitView()
##
##  Assign edge seeds
##
p = mdb.models['standard'].parts['Blank']
e = p.edges
pickedEdges = e.findAt(((-0.025, -0.005, 0.0), ), ((0.025, -0.004, 0.0), ))
p.seedEdgeByNumber(edges=pickedEdges, number=100)
pickedEdges = e.findAt(((-0.05, -0.00425, 0.0), ), ((0.05, -0.00475, 0.0), ))
p.seedEdgeByNumber(edges=pickedEdges, number=4)
##
##  Assign element type
##
elemType1 = mesh.ElemType(elemCode=CPE4R, elemLibrary=STANDARD,
    secondOrderAccuracy=OFF, hourglassControl=ENHANCED, distortionControl=OFF)
elemType2 = mesh.ElemType(elemCode=CPE3, elemLibrary=STANDARD)
f = p.faces
faces = f
pickedRegions =(faces, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
##
##  Use structured meshing
##
f = p.faces
pickedRegions = f
p.setMeshControls(regions=pickedRegions, technique=STRUCTURED)
##
##  Generate mesh
##
p.generateMesh()
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
   geometryEdgesInShaded=OFF, datumPoints=OFF, datumAxes=OFF, datumPlanes=OFF,

    datumCoordSystems=OFF)

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
##
##  Create job
##
mdb.Job(name='Channel', model='standard',
    description='Analysis of the forming of a channel-static')
##
##  Save model database
##
mdb.saveAs('Channel')
##

##
##  Create explicit dynamics model
##
mdb.Model(name='explicit', objectToCopy=mdb.models['standard'])
#: The model "explicit" has been created.

p = mdb.models['explicit'].parts['Blank']
```

```
session.viewports['Viewport: 1'].setValues(displayedObject=p)

##
##  add density
##
mdb.models['explicit'].materials['Steel'].Density(table=((7800.0, ), ))

p = mdb.models['explicit'].parts['Holder']
session.viewports['Viewport: 1'].setValues(displayedObject=p)

##
##  add point mass to holder
##
r = p.referencePoints
refPoints=(r[2], )
region = regionToolset.Region(referencePoints=refPoints)
mdb.models['explicit'].parts['Holder'].engineeringFeatures.PointMassInertia(
    name='mass', region=region, mass=0.1, alpha=0.0,
    composite=0.0)

a = mdb.models['explicit'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

##
##  delete all but first step
##
del mdb.models['explicit'].steps['Move punch']

##
##  define set containing blank
##
f1 = a.instances['Blank-1'].faces
faces1 = f1
a.Set(faces=faces1, name='blank')

##
##  replace the remaining static step with an explicit dynamics step
##
mdb.models['explicit'].ExplicitDynamicsStep(name='Holder force',
    previous='Initial', maintainAttributes=True, timePeriod=0.0001)

##
##  create a second explicit dynamics step
##
mdb.models['explicit'].ExplicitDynamicsStep(name='Move punch',
    previous='Holder force', timePeriod=0.007,
    description='Apply punch stroke')

##
##  define mass scaling
##
regionDef=mdb.models['explicit'].rootAssembly.sets['blank']
mdb.models['explicit'].steps['Holder force'].setValues(massScaling=((
    SEMI_AUTOMATIC, regionDef, AT_BEGINNING, 5.0, 0.0, None, 0, 0, 0.0,
    0.0, 0, None), ))

##
##  request RF2, U2 history output for punch reference point
##
regionDef=mdb.models['explicit'].rootAssembly.sets['RefPunch']
mdb.models['explicit'].HistoryOutputRequest(name='H-Output-2',
```

```
    createStepName='Holder force', variables=('U2', 'RF2'),
    region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE,
    filter=ANTIALIASING)

##
##  define smooth step amplitude curves
##
mdb.models['explicit'].SmoothStepAmplitude(name='smooth1', timeSpan=STEP,
    data=((0.0, 0.0), (0.0001, 1.0)))
mdb.models['explicit'].SmoothStepAmplitude(name='smooth2', timeSpan=STEP,
    data=((0.0, 0.0), (0.007, 1.0)))
##
##  use smooth step amplitude for holder force
##
mdb.models['explicit'].loads['RefHolderForce'].setValues(amplitude='smooth1')

##
##  use smooth step amplitude for punch bc
##
mdb.models['explicit'].boundaryConditions['RefPunchBC'].setValuesInStep(
    stepName='Move punch', u2=-0.030, amplitude='smooth2')

elemType1 = mesh.ElemType(elemCode=CPE4R, elemLibrary=EXPLICIT,
    secondOrderAccuracy=OFF, hourglassControl=ENHANCED, distortionControl=OFF)
elemType2 = mesh.ElemType(elemCode=CPE3, elemLibrary=EXPLICIT)

p = mdb.models['explicit'].parts['Blank']
f = p.faces
faces = f
pickedRegions =(faces, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))

a.regenerate()
p.generateMesh()

##
##  create job
##
mdb.Job(name='expChannel', model='explicit',
    description='Analysis of the forming of a channel-quasistatic')

a = mdb.models['standard'].rootAssembly
a.regenerate()
a = mdb.models['explicit'].rootAssembly
a.regenerate()

mdb.save()
```

gsi_channel_caemodel_spring.py

```
#
# Getting Started with Abaqus
#
# Script for springback portion of channel example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
```

```
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
openMdb('Channel.cae')
a = mdb.models['explicit'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

##
##  Springback analysis model
##

mdb.Model(name='import', objectToCopy=mdb.models['explicit'])

a = mdb.models['import'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

a.deleteFeatures(('Punch-1', 'Holder-1', 'Die-1', ))
del mdb.models['import'].rootAssembly.sets['RefDie']
del mdb.models['import'].rootAssembly.sets['RefHolder']
del mdb.models['import'].rootAssembly.sets['RefPunch']
del mdb.models['import'].rootAssembly.surfaces['BlankBot']
del mdb.models['import'].rootAssembly.surfaces['BlankTop']
del mdb.models['import'].rootAssembly.surfaces['DieSurf']
del mdb.models['import'].rootAssembly.surfaces['HolderSurf']
del mdb.models['import'].rootAssembly.surfaces['PunchSurf']
mdb.models['import'].interactions.delete(
    ('Die-Blank', 'Holder-Blank', 'Punch-Blank', ))
del mdb.models['import'].interactionProperties['Fric']
del mdb.models['import'].interactionProperties['NoFric']
del mdb.models['import'].steps['Move punch']
del mdb.models['import'].steps['Holder force']

region = a.sets['Center']
mdb.models['import'].XsymmBC(name='BC-1', createStepName='Initial',
    region=region)

a = mdb.models['import'].rootAssembly
n1 = a.instances['Blank-1'].nodes
nodes1 = n1[2:3]
a.Set(nodes=nodes1, name='MidLeft')

region = a.sets['MidLeft']
mdb.models['import'].VelocityBC(name='BC-2', createStepName='Initial',
    region=region, v1=UNSET, v2=SET, vr3=UNSET, amplitude=UNSET,
    distributionType=UNIFORM, localCsys=None)

mdb.models['import'].StaticStep(name='springback', previous='Initial',
    initialInc=0.1, nlgeom=ON, stabilizationMagnitude=0.0002,
    stabilizationMethod=DISSIPATED_ENERGY_FRACTION, adaptiveDampingRatio=0.0)

instances=(a.instances['Blank-1'], )
mdb.models['import'].InitialState(updateReferenceConfiguration=OFF,
    fileName='expChannel', endStep=LAST_STEP, endIncrement=STEP_END,
    name='Field-1', createStepName='Initial', instances=instances)

mdb.Job(name='springback', model='import',
    description='Analysis of the forming of a channel-springback')
```

```
a = mdb.models['import'].rootAssembly
a.regenerate()

mdb.save()
```

Shearing of a lap joint

This Python script and additional files can be used to create models for examples contained in this guide.

- gsi_lap_joint_caemodel.py
- lap-joint-alum.txt
- lap-joint-titanium.txt

Fetching the Python script

The scripts are included in Abaqus release; you can extract them from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File**->**Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gsi_lap_joint_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Lap joint example
#
from abaqus import *
from abaqusConstants import *
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
from caeModules import *
Mdb()

session.viewports['Viewport: 1'].setValues(displayedObject=None)
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=30.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(0.0, 0.0), point2=(30.0, 10.0))
p = mdb.models['Model-1'].Part(name='plate', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['plate']
p.BaseSolidExtrude(sketch=s, depth=1.5)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['plate']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

p = mdb.models['Model-1'].parts['plate']
```

```
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(10.0, 3.333333,
    1.5)), sketchUpEdge=e.findAt(coordinates=(30.0, 7.5, 1.5)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(15.0, 5.0, 1.5))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=63.31, gridSpacing=1.58, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['plate']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.CircleByCenterPerimeter(center=(-9.48, -5.0), point1=(-9.48, -7.9))
s.CoincidentConstraint(entity1=v.findAt((-9.48, -5.0)), entity2=g.findAt((0.0,

    -5.0)))
s.HorizontalDimension(vertex1=v.findAt((-15.0, -5.0)), vertex2=v.findAt((
   -9.48, -5.0)), textPoint=(-10.4236354827881, -9.93480682373047), value=7.5)
s.RadialDimension(curve=g.findAt((-7.5, -2.1)), textPoint=(-2.11909103393555,
    -9.19041156768799), radius=2.5)
p.CutExtrude(sketchPlane=f.findAt(coordinates=(10.0, 3.333333, 1.5)),
    sketchUpEdge=e.findAt(coordinates=(30.0, 7.5, 1.5)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s,
    flipExtrudeDirection=OFF)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']

c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=20.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.ConstructionLine(point1=(0.0, -10.0), point2=(0.0, 10.0))
s.FixedConstraint(entity=g.findAt((0.0, 0.0)))
s.Line(point1=(0.0, 2.5), point2=(0.0, -2.52870798110962))
s.VerticalConstraint(entity=g.findAt((0.0, -2.277273)), addUndoState=False)
s.ParallelConstraint(entity1=g.findAt((0.0, 2222231.04)), entity2=g.findAt((
    0.0, -2.277273)), addUndoState=False)
s.CoincidentConstraint(entity1=v.findAt((0.0, 2.5)), entity2=g.findAt((0.0,
    2222231.04)), addUndoState=False)
s.CoincidentConstraint(entity1=v.findAt((0.0, -2.528708)), entity2=g.findAt((
    0.0, 2222231.04)), addUndoState=False)
s.Line(point1=(0.0, -2.52870798110962), point2=(2.625, -2.52870798110962))
s.HorizontalConstraint(entity=g.findAt((1.3125, -2.528708)),
    addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((0.0, -2.277273)),
    entity2=g.findAt((1.3125, -2.528708)), addUndoState=False)
s.Line(point1=(2.625, -2.52870798110962), point2=(2.625, -1.625))
s.VerticalConstraint(entity=g.findAt((2.625, -2.076854)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((1.3125, -2.528708)),
    entity2=g.findAt((2.625, -2.076854)), addUndoState=False)
s.Line(point1=(2.625, -1.625), point2=(1.5, -1.625))
s.HorizontalConstraint(entity=g.findAt((2.0625, -1.625)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((2.625, -2.076854)),
    entity2=g.findAt((2.0625, -1.625)), addUndoState=False)
s.Line(point1=(1.5, -1.625), point2=(1.5, 1.75))
s.VerticalConstraint(entity=g.findAt((1.5, 0.0625)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((2.0625, -1.625)),
    entity2=g.findAt((1.5, 0.0625)), addUndoState=False)
s.Line(point1=(1.5, 1.75), point2=(2.625, 1.75))
s.HorizontalConstraint(entity=g.findAt((2.0625, 1.75)), addUndoState=False)
```

```
s.PerpendicularConstraint(entity1=g.findAt((1.5, 0.0625)), entity2=g.findAt((
    2.0625, 1.75)), addUndoState=False)
s.Line(point1=(2.625, 1.75), point2=(2.625, 2.5))
s.VerticalConstraint(entity=g.findAt((2.625, 2.125)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((2.0625, 1.75)), entity2=g.findAt((
    2.625, 2.125)), addUndoState=False)
s.Line(point1=(2.625, 2.5), point2=(0.0, 2.5))
s.HorizontalConstraint(entity=g.findAt((1.3125, 2.5)), addUndoState=False)
s.PerpendicularConstraint(entity1=g.findAt((2.625, 2.125)), entity2=g.findAt((
    1.3125, 2.5)), addUndoState=False)
s.EqualLengthConstraint(entity1=g.findAt((1.3125, -2.528708)),
    entity2=g.findAt((1.3125, 2.5)))
s.ObliqueDimension(vertex1=v.findAt((0.0, -2.528708)), vertex2=v.findAt((
    2.625, -2.528708)), textPoint=(0.781737446784973, -3.08692169189453),
    value=4.0)
s.DistanceDimension(entity1=g.findAt((0.0, 2222231.04)), entity2=g.findAt((
    1.5, 0.0625)), textPoint=(1.5, -1.02153098583221), value=2.5)
session.viewports['Viewport: 1'].view.fitView()
s.ObliqueDimension(vertex1=v.findAt((2.5, -1.625)), vertex2=v.findAt((2.5,
    1.75)), textPoint=(3.08898115158081, 0.803566813468933), value=3.0)
s.EqualLengthConstraint(entity1=g.findAt((4.0, 2.125)), entity2=g.findAt((4.0,

    -1.889354)))
s.ObliqueDimension(vertex1=v.findAt((4.0, 1.573764)), vertex2=v.findAt((4.0,
    2.323764)), textPoint=(4.55753326416016, 1.82851922512054), value=1.5)
session.viewports['Viewport: 1'].view.fitView()
s.move(vector=(0.0, -1.07376400629679), objectList=(g.findAt((0.0,
    1925933.568)), g.findAt((0.0, -2.866236)), g.findAt((2.0, -2.926236)),
    g.findAt((4.0, -2.176236)), g.findAt((3.25, -1.426236)), g.findAt((2.5,
    0.073764)), g.findAt((3.25, 1.573764)), g.findAt((4.0, 2.323764)),
    g.findAt((2.0, 3.073764))))
session.viewports['Viewport: 1'].view.fitView()
p = mdb.models['Model-1'].Part(name='rivet', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['rivet']
p.BaseSolidRevolve(sketch=s, angle=180.0, flipRevolveDirection=OFF)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['rivet']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

p = mdb.models['Model-1'].parts['rivet']
c, d, e, f, v = p.cells, p.datums, p.edges, p.faces, p.vertices

p.Round(radius=0.75, edgeList=(e.findAt(coordinates=(2.828427, 2.0, 2.828427)),

    ))
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums

p.Chamfer(length=0.75, edgeList=(e.findAt(coordinates=(2.828427, -4.0,
    2.828427)), ))
c, f, e, v, d  = p.cells, p.faces, p.edges, p.vertices, p.datums


a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
a.DatumCsysByDefault(CARTESIAN)

p = mdb.models['Model-1'].parts['plate']
a.Instance(name='plate-1', part=p, dependent=ON)
```

```
a.Instance(name='plate-2', part=p, dependent=ON)

p1 = a.instances['plate-2']
p1.translate(vector=(33.0, 0.0, 0.0))

a.rotate(
    instanceList=('plate-2', ),
    axisPoint=(33.0, 10.0, 1.5),
    axisDirection=(0.0, -1.0, 0.0),
    angle=180.0)

f1 = a.instances['plate-2'].faces
f2 = a.instances['plate-1'].faces
a.FaceToFace(
    movablePlane=f1.findAt(coordinates=(25.633485, 4.98924, 3.0)),
    fixedPlane=f2.findAt(coordinates=(22.0, 6.666667, 0.0)),
    flip=ON,
    clearance=0.0)
a.Coaxial(
    movableAxis=f1.findAt(coordinates=(23.005337, 0.163274, -0.5)),
    fixedAxis=f2.findAt(coordinates=(9.994663, 0.163274, 0.5)),
    flip=ON)

p = mdb.models['Model-1'].parts['rivet']
a.Instance(name='rivet-1', part=p, dependent=ON)
p1 = a.instances['rivet-1']
p1.translate(vector=(34.8, 0.0, 0.0))

f1 = a.instances['rivet-1'].faces
f2 = a.instances['plate-2'].faces
a.Coaxial(
    movableAxis=f1.findAt(coordinates=(32.305337, -0.5, 0.163274)),
    fixedAxis=f2.findAt(coordinates=(5.005337, 0.163274, -0.5)),
    flip=ON)
a.rotate(
    instanceList=('rivet-1', ),
    axisPoint=(7.5, 0.0, 1.5),
    axisDirection=(0.0, 0.0, 1.5),
    angle=180.0)

p = mdb.models['Model-1'].parts['plate']
c = p.cells
e = p.edges

pickedRegions = c.findAt(((3.333333, 0.0, 1.0), ))
p.setMeshControls(regions=pickedRegions, allowMapped=True)

p.seedPart(size=1.2, deviationFactor=0.1)
import re, uti
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=1.75, deviationFactor=0.05)


elemType1 = mesh.ElemType(elemCode=C3D8I, elemLibrary=STANDARD,
    secondOrderAccuracy=OFF, distortionControl=DEFAULT)
elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)
cells = c
pickedRegions =(cells, )
p.setElementType(
```

```
    regions=pickedRegions,
    elemTypes=(elemType1, elemType2, elemType3))
p.generateMesh()

p = mdb.models['Model-1'].parts['rivet']
c = p.cells
e = p.edges

p.seedPart(size=0.5, deviationFactor=0.1)
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedPart(size=1., deviationFactor=0.1)

pickedRegions = c.findAt(((2.139505, -4.0, 0.241064), ))
p.setMeshControls(
    regions=pickedRegions,
    elemShape=HEX_DOMINATED,
    technique=SWEEP,
    algorithm=ADVANCING_FRONT,
    allowMapped=True)

elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=STANDARD,
    kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT, distortionControl=DEFAULT)
elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=STANDARD)
cells = c
pickedRegions =(cells, )
p.setElementType(
    regions=pickedRegions,
    elemTypes=(elemType1, elemType2, elemType3))
p.generateMesh()

a.regenerate()

f1 = a.instances['plate-2'].faces
faces1 = f1.findAt(((11.666667, 0.0, -1.0), ), ((-1.666667, 0.0, -0.5), ))
f2 = a.instances['plate-1'].faces
faces2 = f2.findAt(((3.333333, 0.0, 1.0), ), ((16.666667, 0.0, 0.5), ))
f3 = a.instances['rivet-1'].faces
faces3 = f3.findAt(((4.0, 0.0, -1.75), ), ((9.416667, 0.0, -2.5), ))
a.Set(faces=faces1+faces2+faces3, name='symm')



f1 = a.instances['plate-1'].faces
faces1 = f1.findAt(((30.0, 6.666667, 1.0), ))
a.Set(faces=faces1, name='pull')

f1 = a.instances['plate-2'].faces
faces1 = f1.findAt(((-15.0, 6.666667, -1.0), ))
a.Set(faces=faces1, name='fix')

v1 = a.instances['plate-2'].vertices
verts1 = v1.findAt(((-15.0, 0.0, -1.5), ))
a.Set(vertices=verts1, name='corner')

mdb.models['Model-1'].Material(name='aluminum')
mdb.models['Model-1'].materials['aluminum'].Density(table=((0.0028, ), ))
mdb.models['Model-1'].materials['aluminum'].Elastic(table=((71700.0, 0.33), ))
mdb.models['Model-1'].materials['aluminum'].Plastic(table=(
```

```
    (350.0, 0.0),
    (368.71, 0.001),
    (376.5, 0.002),
    (391.98, 0.005),
    (403.15, 0.008),
    (412.36, 0.011),
    (422.87, 0.015),
    (444.17, 0.025),
    (461.5, 0.035),
    (507.9, 0.07),
    (581.5, 0.15),
    (649.17, 0.25),
    (704.22, 0.35),
    (728.78, 0.4),
    (751.85, 0.45),
    (773.68, 0.5),
    (794.44, 0.55),
    (814.28, 0.6)))

mdb.models['Model-1'].Material(name='titanium')
mdb.models['Model-1'].materials['titanium'].Density(table=((0.0044, ), ))
mdb.models['Model-1'].materials['titanium'].Elastic(table=((112000.0, 0.34),
))
mdb.models['Model-1'].materials['titanium'].Plastic(table=(
    (907.0, 0.0),
    (934.86, 0.001),
    (944.28, 0.002),
    (961.77, 0.005),
    (973.73, 0.008),
    (983.28, 0.011),
    (993.89, 0.015),
    (1014.7, 0.025),
    (1023.3, 0.03),
    (1051.1, 0.05),
    (1099.8, 0.1),
    (1129.0, 0.14),
    (1164.9, 0.2),
    (1190.2, 0.25),
    (1212.8, 0.3)))

mdb.models['Model-1'].HomogeneousSolidSection(
    name='plate',
    material='aluminum',
    thickness=1.0)

p = mdb.models['Model-1'].parts['plate']
c = p.cells
cells = c
region = regionToolset.Region(cells=cells)
p.SectionAssignment(
    region=region,
    sectionName='plate',
    offset=0.0)

mdb.models['Model-1'].HomogeneousSolidSection(
    name='rivet',
    material='titanium',
    thickness=1.0)

p = mdb.models['Model-1'].parts['rivet']
c = p.cells
```

```
cells = c
region = regionToolset.Region(cells=cells)
p.SectionAssignment(
    region=region,
    sectionName='rivet',
    offset=0.0)

a.regenerate()

session.viewports['Viewport: 1'].view.fitView()

mdb.models['Model-1'].StaticStep(
    name='Step-1',
    previous='Initial',
    maxNumInc=100,
    stabilizationMethod=NONE,
    initialInc=0.05,
    nlgeom=ON)

region = a.sets['fix']
mdb.models['Model-1'].DisplacementBC(
    name='Fix',
    createStepName='Step-1',
    region=region,
    u1=0.0)

region = a.sets['pull']
mdb.models['Model-1'].DisplacementBC(
    name='Pull',
    createStepName='Step-1',
    region=region,
    u1=2.5)

region = a.sets['symm']
mdb.models['Model-1'].DisplacementBC(
    name='Symmetry',
    createStepName='Step-1',
    region=region,
    u2=0.0)

region = a.sets['corner']
mdb.models['Model-1'].DisplacementBC(
    name='RB',
    createStepName='Step-1',
    region=region,
    u3=0.0)

mdb.models['Model-1'].ContactProperty('fric')
mdb.models['Model-1'].interactionProperties['fric'].TangentialBehavior(
    formulation=PENALTY,
    directionality=ISOTROPIC,
    slipRateDependency=OFF,
    pressureDependency=OFF,
    temperatureDependency=OFF,
    dependencies=0,
    table=((0.05, ), ),
    shearStressLimit=None,
    maximumElasticSlip=FRACTION,
    fraction=0.005,
    elasticSlipStiffness=None)
```

```
mdb.models['Model-1'].ContactStd(name='Int-1', createStepName='Initial')
mdb.models['Model-1'].interactions['Int-1'].includedPairs.setValuesInStep(
    stepName='Initial',
    useAllstar=ON)
mdb.models['Model-1'].interactions['Int-1'].contactPropertyAssignments.appendInStep(

    stepName='Initial',
    assignments=((GLOBAL, SELF, 'fric'), ))

mdb.Job(
    name='lap_joint',
    model='Model-1',
    type=ANALYSIS)

mdb.saveAs(pathName='lap_joint')
```

lap-joint-alum.txt

```
0.0000, 0.000000000
350.00, 0.004881450
368.71, 0.006142399
376.50, 0.007251046
391.98, 0.010466946
403.15, 0.013622734
412.36, 0.016751185
422.87, 0.020897768
444.17, 0.031194840
461.50, 0.041436541
507.90, 0.077083682
581.50, 0.158110181
649.17, 0.259053975
704.22, 0.359821757
728.78, 0.410164296
751.85, 0.460486053
773.68, 0.510790516
794.44, 0.561080056
814.28, 0.611356764
```

lap-joint-titanium.txt

```
0.0000, 0.000000000
907.00, 0.008098214
934.86, 0.009346964
944.28, 0.010431071
961.77, 0.013587232
973.73, 0.016694018
983.28, 0.019779286
993.89, 0.023874018
1014.7, 0.034059821
1023.3, 0.039136607
1051.1, 0.059384821
1099.8, 0.109819643
1129.0, 0.150080357
1164.9, 0.210400893
1190.2, 0.260626786
1212.8, 0.310828571
```

Circuit board drop test

These Python scripts can be used to create models for examples contained in this guide.

- gxi_circuit_caemodel.py
- drop-test-foam.txt

Fetching the Python script

The scripts are included in the Abaqus release; you can extract them from the compressed archive files using the **abaqus fetch** utility.

- To fetch a script, enter the following command at the operating system prompt:

  ```
  abaqus fetch job=file_name
  ```

  where *file_name* includes the extension `.py`.
- To run a script in Abaqus/CAE, do the following:

  1. From the main menu bar, select**File**->**Run Script**.

     The **Run Script** dialog box appears.
  2. Choose the file from the list of available scripts, and click **OK**.

gxi_circuit_caemodel.py

```
#
# Getting Started with Abaqus: Interactive Edition
#
# Script for circuit board drop test example
#
from abaqus import *
from abaqusConstants import *
backwardCompatibility.setValues(includeDeprecated=True, reportDeprecated=False)

session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()
session.journalOptions.setValues(replayGeometry=COORDINATE,
    recoverGeometry=COORDINATE)
from caeModules import *
Mdb()

session.viewports['Viewport: 1'].setValues(displayedObject=None)
mdb.models.changeKey(fromName='Model-1', toName='circuitBoard')

s = mdb.models['circuitBoard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.1)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=0.1, gridSpacing=0.002, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.002,
    decimalPlaces=3)
s.rectangle(point1=(-0.01, -0.012), point2=(0.01, 0.012))
p = mdb.models['circuitBoard'].Part(name='Packaging', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['circuitBoard'].parts['Packaging']
```

```
p.BaseSolidExtrude(sketch=s, depth=0.11)
s.unsetPrimaryObject()
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['circuitBoard'].sketches['__profile__']

p = mdb.models['circuitBoard'].parts['Packaging']
session.viewports['Viewport: 1'].setValues(displayedObject=p)

p = mdb.models['circuitBoard'].parts['Packaging']
f, e = p.faces, p.edges
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-0.003333, -0.004,

    0.11)), sketchUpEdge=e.findAt(coordinates=(0.01, 0.006, 0.11)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.11))
s = mdb.models['circuitBoard'].ConstrainedSketch(name='__profile__',
    sheetSize=0.228, gridSpacing=0.005, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.sketchOptions.setValues(decimalPlaces=3)
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['circuitBoard'].parts['Packaging']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.ConstructionLine(point1=(0.0, 0.0), angle=90.0)
s.VerticalConstraint(entity=g.findAt((0.0, 0.5)), addUndoState=False)
s.FixedConstraint(entity=g.findAt((0.0, 0.5)))
session.viewports['Viewport: 1'].view.setValues(nearPlane=0.0543671,
    farPlane=0.180597, width=0.0480393, height=0.0404284)
s.rectangle(point1=(-0.0025, 0.01625), point2=(0.0025, 0.005))
s.SymmetryConstraint(entity1=g.findAt((-0.0025, 0.010625)), entity2=g.findAt((
    0.0025, 0.010625)), symmetryAxis=g.findAt((0.0, 0.5)))
s.ObliqueDimension(vertex1=v.findAt((-0.0025, 0.005)), vertex2=v.findAt((
    0.0025, 0.005)), textPoint=(0.0015479710418731, 0.00119286333210766),
    value=0.002)
s.VerticalDimension(vertex1=v.findAt((0.01, 0.012)), vertex2=v.findAt((0.001,
    0.005)), textPoint=(0.013351246714592, 0.00686702458187938), value=0.012)
p = mdb.models['circuitBoard'].parts['Packaging']
f1, e1 = p.faces, p.edges
p.CutExtrude(sketchPlane=f1.findAt(coordinates=(-0.003333, -0.004, 0.11)),
    sketchUpEdge=e1.findAt(coordinates=(0.01, 0.006, 0.11)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s,
    flipExtrudeDirection=OFF)
s.unsetPrimaryObject()
del mdb.models['circuitBoard'].sketches['__profile__']

p.DatumPointByMidPoint(point1=p.InterestingPoint(edge=e.findAt(coordinates=(
    0.0005, 0.0, 0.11)), rule=MIDDLE), point2=p.InterestingPoint(
    edge=e.findAt(coordinates=(-0.0005, 0.0, 0.0)), rule=MIDDLE))

s = mdb.models['circuitBoard'].Sketch(name='__profile__', sheetSize=0.5)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=0.5, gridSpacing=0.01, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.01,
    decimalPlaces=3)
s.rectangle(point1=(0.0, 0.0), point2=(0.1, 0.15))
session.viewports['Viewport: 1'].view.fitView()
p = mdb.models['circuitBoard'].Part(name='Board', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['circuitBoard'].parts['Board']
p.BaseShell(sketch=s)
p = mdb.models['circuitBoard'].parts['Board']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['circuitBoard'].sketches['__profile__']
```

```
p = mdb.models['circuitBoard'].parts['Board']
v = p.vertices
p.DatumPointByOffset(point=v.findAt(coordinates=(0.0, 0.0, 0.0)), vector=(
    0.01, 0.135, 0.0))
p.DatumPointByOffset(point=v.findAt(coordinates=(0.0, 0.0, 0.0)), vector=(
    0.07, 0.09, 0.0))
p.DatumPointByOffset(point=v.findAt(coordinates=(0.0, 0.0, 0.0)), vector=(
    0.08, 0.03, 0.0))

s = mdb.models['circuitBoard'].Sketch(name='__profile__', sheetSize=0.5)
g, v, d = s.geometry, s.vertices, s.dimensions
s.sketchOptions.setValues(sheetSize=0.5, gridSpacing=0.01, grid=ON,
    gridFrequency=2, constructionGeometry=ON, dimensionTextHeight=0.01,
    decimalPlaces=3)
s.rectangle(point1=(-0.1, -0.1), point2=(0.1, 0.1))
p = mdb.models['circuitBoard'].Part(name='Floor', dimensionality=THREE_D,
    type=DISCRETE_RIGID_SURFACE)
p = mdb.models['circuitBoard'].parts['Floor']
p.BaseShell(sketch=s)
p = mdb.models['circuitBoard'].parts['Floor']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['circuitBoard'].sketches['__profile__']

p = mdb.models['circuitBoard'].parts['Floor']
p.ReferencePoint(point=(0.0, 0.0, 0.0))
session.viewports['Viewport: 1'].setValues(displayedObject=p)

mdb.models['circuitBoard'].Material('PCB')
mdb.models['circuitBoard'].materials['PCB'].Elastic(table=((45.e9, 0.3), ))
mdb.models['circuitBoard'].materials['PCB'].Density(table=((500.0, ), ))
mdb.models['circuitBoard'].Material('Foam')
mdb.models['circuitBoard'].materials['Foam'].Elastic(table=((3.e6, 0.0), ))
mdb.models['circuitBoard'].materials['Foam'].Density(table=((100.0, ), ))
mdb.models['circuitBoard'].materials['Foam'].CrushableFoam(table=((1.1, 0.1),))
mdb.models['circuitBoard'].materials['Foam'].crushableFoam.CrushableFoamHardening(table=(

    (220000.0, 0.0), (246510.0, 0.1), (272940.0, 0.2), (299020.0, 0.3),
    (324550.0, 0.4), (349350.0, 0.5), (373260.0, 0.6), (396170.0, 0.7),
    (418010.0, 0.8), (438720.0, 0.9), (458270.0, 1.0), (493840.0, 1.2),
    (524840.0, 1.4), (551530.0, 1.6), (574310.0, 1.8), (593590.0, 2.0),
    (629360.0, 2.5), (651990.0, 3.0), (683340.0, 5.0), (688330.0, 10.0)))

mdb.models['circuitBoard'].HomogeneousShellSection(name='BoardSection',
    preIntegrate=OFF, material='PCB', thickness=0.002,
    poissonDefinition=DEFAULT, temperature=GRADIENT, integrationRule=SIMPSON,
    numIntPts=5)

mdb.models['circuitBoard'].HomogeneousSolidSection(name='FoamSection',
    material='Foam', thickness=1.0)

p = mdb.models['circuitBoard'].parts['Board']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
f = p.faces
faces = f
region = regionToolset.Region(faces=faces)
p.SectionAssignment(region=region, sectionName='BoardSection')

p = mdb.models['circuitBoard'].parts['Packaging']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
c = p.cells
```

```
cells = c
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='FoamSection')

p = mdb.models['circuitBoard'].parts['Board']
session.viewports['Viewport: 1'].setValues(displayedObject=p)

e = p.edges
p.DatumCsysByTwoLines(CARTESIAN,
    line1=e.findAt(coordinates=(0.025, 0.0, 0.0)),
    line2=e.findAt(coordinates=(0.1, 0.0375, 0.0)), name='Datum csys-1')

f = p.faces
faces = f
region = regionToolset.Region(faces=faces)
datums = p.datums[6]
p.MaterialOrientation(region=region, localCsys=datums, axis=AXIS_3)

a = mdb.models['circuitBoard'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)

a.DatumCsysByDefault(CARTESIAN)

p = mdb.models['circuitBoard'].parts['Floor']
a.Instance(name='Floor-1', part=p, dependent=ON)

a.DatumPointByCoordinate(coords=(0.0, 0.0, 0.0))
a.DatumPointByCoordinate(coords=(0.5, 0.707, 0.25))
d = a.datums
a.DatumAxisByTwoPoint(point1=d[4], point2=d[5])

p = mdb.models['circuitBoard'].parts['Packaging']
a.Instance(name='Packaging-1', part=p, dependent=ON)
mdb.models['circuitBoard'].rootAssembly.setValues(
   regenerateConstraintsTogether=OFF)

e = a.instances['Packaging-1'].edges
d = a.datums
a.EdgeToEdge(movableAxis=e.findAt(coordinates=(-0.01, -0.012, 0.0275)),
    fixedAxis=d[6], flip=ON)

a.DatumPointByCoordinate(coords=(-0.5, 0.707, -0.5))
d = a.datums
a.DatumPlaneByLinePoint(line=d[6], point=d[10])

f = a.instances['Packaging-1'].faces
d = a.datums
a.FaceToFace(movablePlane=f.findAt(coordinates=(0.0179891354276455,
    0.00658330416135736, 0.012563775694313)), fixedPlane=d[11], flip=ON,
    clearance=0.0)

v = a.instances['Packaging-1'].vertices
r = a.instances['Floor-1'].referencePoints
a.CoincidentPoint(movablePoint=v.findAt(coordinates=(-0.0303476233973587,
    -0.0429115394838653, -0.0151738116986794)), fixedPoint=r[2])

p = a.instances['Packaging-1']
p.ConvertConstraints()

p = a.instances['Floor-1']
p.translate(vector=(0.0, 0.0, -0.0001))
```

```
p = mdb.models['circuitBoard'].parts['Board']
a.Instance(name='Board-1', part=p, dependent=ON)

p = a.instances['Board-1']
p.translate(vector=(0.51, 0.0, 0.0))

f1 = a.instances['Board-1'].faces
f2 = a.instances['Packaging-1'].faces
a.ParallelFace(movablePlane=f1.findAt(coordinates=(0.543333333333333, 0.05,
    0.0), normal=(0.0, 0.0, 1.0)), fixedPlane=f2.findAt(coordinates=(
    0.0272382919150449, 0.0177981609225068, 0.0273814785169062)), flip=OFF)

e1 = a.instances['Board-1'].edges
e2 = a.instances['Packaging-1'].edges
a.ParallelEdge(movableAxis=e1.findAt(coordinates=(0.596604525482204,
    0.137963003192081, 0.0307533566177196)), fixedAxis=e2.findAt(coordinates=(
    0.0431590254312432, 0.0559914769973559, 0.0527683904196429)), flip=OFF)

e1 = a.instances['Board-1'].edges
d1 = a.instances['Packaging-1'].datums
a.CoincidentPoint(fixedPoint=d1[3],
    movablePoint=a.instances['Board-1'].InterestingPoint(
    edge=e1.findAt(coordinates=(0.652343827802905, 0.0916453486569404,
    -0.0804357827095802)), rule=MIDDLE))

d = a.instances['Board-1'].datums
a.ReferencePoint(point=d[2])
a.ReferencePoint(point=d[3])
a.ReferencePoint(point=d[4])

r = a.referencePoints
refPoints1=(r[19], )
a.Set(referencePoints=refPoints1, name='TopChip')
refPoints1=(r[20], )
a.Set(referencePoints=refPoints1, name='MidChip')
refPoints1=(r[21], )
a.Set(referencePoints=refPoints1, name='BotChip')
refPoints1=(r[21], )
a.Set(referencePoints=refPoints1, name='BotChip-all')
refPoints1=(r[21], )
a.Set(referencePoints=refPoints1, name='BotChip-largeInc')

e = a.instances['Board-1'].edges
edges = e.findAt(((0.0153385555521983, 0.0191710250695974,
    0.0232637166281098), ))
a.Set(edges=edges, name='BotBoard')

region=a.sets['TopChip']
mdb.models['circuitBoard'].rootAssembly.engineeringFeatures.PointMassInertia(
    name='MassTopChip', region=region, mass=0.005, alpha=0.0, composite=0.0)

region=a.sets['MidChip']
mdb.models['circuitBoard'].rootAssembly.engineeringFeatures.PointMassInertia(
    name='MassMidChip', region=region, mass=0.005, alpha=0.0, composite=0.0)

region=a.sets['BotChip']
mdb.models['circuitBoard'].rootAssembly.engineeringFeatures.PointMassInertia(
    name='MassBotChip', region=region, mass=0.005, alpha=0.0, composite=0.0)

mdb.models['circuitBoard'].ExplicitDynamicsStep(name='Drop',
```

```
    previous='Initial', timePeriod=0.02, massScaling=PREVIOUS_STEP)

regionDef=mdb.models['circuitBoard'].rootAssembly.sets['TopChip']
mdb.models['circuitBoard'].HistoryOutputRequest(name='H-Output-TopChip',
    createStepName='Drop', variables=('A3', 'V3', 'U3' ),
    timeInterval=0.00007, region=regionDef)

mdb.models['circuitBoard'].HistoryOutputRequest('H-Output-MidChip',
    mdb.models['circuitBoard'].historyOutputRequests['H-Output-TopChip'])
regionDef=mdb.models['circuitBoard'].rootAssembly.sets['MidChip']
mdb.models['circuitBoard'].historyOutputRequests['H-Output-MidChip'].setValues(

    region=regionDef)

mdb.models['circuitBoard'].HistoryOutputRequest('H-Output-BotChip',
    mdb.models['circuitBoard'].historyOutputRequests['H-Output-MidChip'])
regionDef=mdb.models['circuitBoard'].rootAssembly.sets['BotChip']
mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip'].setValues(

    region=regionDef)

mdb.models['circuitBoard'].HistoryOutputRequest('H-Output-BotChip-Antialiasing',

    mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip'])
mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip-Antialiasing'].setValues(

    filter=ANTIALIASING)

mdb.models['circuitBoard'].HistoryOutputRequest('H-Output-BotChip-all',
    mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip'])
regionDef=mdb.models['circuitBoard'].rootAssembly.sets['BotChip-all']
mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip-all'].setValues(

    frequency=1, region=regionDef)

mdb.models['circuitBoard'].HistoryOutputRequest('H-Output-BotChip-largeInc',
    mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip'])
regionDef=mdb.models['circuitBoard'].rootAssembly.sets['BotChip-largeInc']
mdb.models['circuitBoard'].historyOutputRequests['H-Output-BotChip-largeInc'].setValues(

    timeInterval=0.0007, region=regionDef, filter=ANTIALIASING)


regionDef=mdb.models['circuitBoard'].rootAssembly.sets['BotBoard']
mdb.models['circuitBoard'].HistoryOutputRequest(name='H-Output-BotBoard',
    createStepName='Drop', variables=('LE11', 'LE22', 'LE12', 'LEP'),
    timeInterval=0.00007, region=regionDef, sectionPoints=(5, ),
filter=ANTIALIASING)

mdb.models['circuitBoard'].ContactProperty('Fric')

mdb.models['circuitBoard'].interactionProperties['Fric'].TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC, slipRateDependency=OFF,
   pressureDependency=OFF, temperatureDependency=OFF, dependencies=0, table=((

    0.3, ), ), shearStressLimit=None, maximumElasticSlip=FRACTION,
    fraction=0.005, elasticSlipStiffness=None)

mdb.models['circuitBoard'].ContactExp(name='All', createStepName='Drop')
mdb.models['circuitBoard'].interactions['All'].includedPairs.setValuesInStep(
    stepName='Drop', useAllstar=ON)
```

```
mdb.models['circuitBoard'].interactions['All'].contactPropertyAssignments.appendInStep(
    stepName='Drop', assignments=((GLOBAL, SELF, 'Fric'), ))
s = a.instances['Board-1'].faces
side12Faces = s
a.Surface(side12Faces=side12Faces, name='Board')
region1=a.surfaces['Board']
region2=a.sets['TopChip']
mdb.models['circuitBoard'].Tie(name='TopChip', main=region1, secondary=region2,
    positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=OFF)
region1=a.surfaces['Board']
region2=a.sets['MidChip']
mdb.models['circuitBoard'].Tie(name='MidChip', main=region1, secondary=region2,
    positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=OFF)
region1=a.surfaces['Board']
region2=a.sets['BotChip']
mdb.models['circuitBoard'].Tie(name='BotChip', main=region1, secondary=region2,
    positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=OFF)
r = a.instances['Floor-1'].referencePoints
refPoints1=(r[2], )
a.Set(referencePoints=refPoints1, name='floorRefPt')
region = a.sets['floorRefPt']
mdb.models['circuitBoard'].EncastreBC(name='BC-1', createStepName='Initial',
    region=region)
c = a.instances['Packaging-1'].cells
cells = c
f = a.instances['Packaging-1'].faces
faces1 = f
e = a.instances['Packaging-1'].edges
edges1 = e
v = a.instances['Packaging-1'].vertices
verts = v
f = a.instances['Board-1'].faces
faces2 = f
e = a.instances['Board-1'].edges
edges2 = e
r = a.referencePoints
refPoints=(r[19], r[20], r[21], )
region = regionToolset.Region(vertices=verts, edges=edges1+edges2,
    faces=faces1+faces2, cells=cells, referencePoints=refPoints)
mdb.models['circuitBoard'].Velocity(name='Field-1', region=region,
    velocity1=0.0, velocity2=0.0, velocity3=-4.43, omega=0.0)
p = mdb.models['circuitBoard'].parts['Board']
e = p.edges
pickedEdges = e.findAt(((0.025, 0.0, 0.0), ), ((0.1, 0.0375, 0.0), ), ((0.0,
    0.1125, 0.0), ), ((0.075, 0.15, 0.0), ))
p.seedEdgeByNumber(edges=pickedEdges, number=10)
f = p.faces
```

```
faces = f
p.setMeshControls(regions=faces, technique=SWEEP)

elemType1 = mesh.ElemType(elemCode=S4R, elemLibrary=EXPLICIT,
    secondOrderAccuracy=OFF, hourglassControl=RELAX_STIFFNESS)
elemType2 = mesh.ElemType(elemCode=S3R, elemLibrary=EXPLICIT)
regions =(faces, )
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2))
p.generateMesh()
##
p = mdb.models['circuitBoard'].parts['Floor']
p.seedPart(size=1.0)

elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
f = p.faces
faces = f
regions =(faces, )
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2))
p.generateMesh()
##
p = mdb.models['circuitBoard'].parts['Packaging']
e = p.edges

edges =(e.findAt(coordinates=(0.0005, 0.0, 0.11)), )
p.seedEdgeByNumber(edges=edges, number=1)

edges = e.findAt(((0.001, 0.009, 0.11), ), ((-0.001, 0.003, 0.11), ), ((
    -0.00775, 0.012, 0.11), ), ((0.00325, 0.012, 0.11), ))
p.seedEdgeByNumber(edges=edges, number=3)

edges = e.findAt(((0.01, 0.006, 0.11), ))
p.seedEdgeByNumber(edges=edges, number=6)

edges = e.findAt(((0.005, -0.012, 0.11), ))
p.seedEdgeByNumber(edges=edges, number=7)

edges = e.findAt(((0.01, 0.012, 0.0275), ))
p.seedEdgeByNumber(edges=edges, number=15)
import re, uti
pat = re.compile('Student|Learning')
if re.search(pat,uti.getProductVersion()):
   p.seedEdgeByNumber(edges=edges, number=12)

elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
    kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
    hourglassControl=ENHANCED)
elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
c = p.cells
cells = c
regions =(cells, )
p.setElementType(regions=regions, elemTypes=(elemType1, elemType2, elemType3))
pickedRegions = c
p.setMeshControls(regions=pickedRegions, algorithm=MEDIAL_AXIS)
p.generateMesh()
##

session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
```

```
mdb.Job(name='Circuit', model='circuitBoard', type=ANALYSIS,
    explicitPrecision=DOUBLE, description='Circuit board drop test')
session.viewports['Viewport: 1'].view.fitView()

a.regenerate()

session.viewports['Viewport: 1'].setValues(displayedObject=a)
session.viewports['Viewport: 1'].view.setValues(session.views['Bottom'])
session.viewports['Viewport: 1'].assemblyDisplay.geometryOptions.setValues(
    datumPoints=OFF, datumAxes=OFF, datumPlanes=OFF, datumCoordSystems=OFF)
session.viewports['Viewport: 1'].view.fitView()

mdb.saveAs('Circuit')
```

drop-test-foam.txt

```
0.22000E6,  0.0
0.24651E6,  0.1
0.27294E6,  0.2
0.29902E6,  0.3
0.32455E6,  0.4
0.34935E6,  0.5
0.37326E6,  0.6
0.39617E6,  0.7
0.41801E6,  0.8
0.43872E6,  0.9
0.45827E6,  1.0
0.49384E6,  1.2
0.52484E6,  1.4
0.55153E6,  1.6
0.57431E6,  1.8
0.59359E6,  2.0
0.62936E6,  2.5
0.65199E6,  3.0
0.68334E6,  5.0
0.68833E6, 10.0
```

Creating and Analyzing a Simple Model in Abaqus/CAE

The following section is a basic tutorial for the experienced Abaqus user. It leads you through the Abaqus/CAE modeling process by visiting each of the modules and showing you the basic steps to create and analyze a simple model.

To illustrate each of the steps, you will first create a model of a steel cantilever beam and load its top surface (see *Figure 1* in *Summary*). You will then analyze the beam and plot the resulting stresses and displacements. The entire tutorial takes approximately 90 minutes to complete.

In this section:

- *Understanding Abaqus/CAE modules*
- *Understanding the Model Tree*
- *Creating a part*
- *Creating a material*
- *Defining and assigning section properties*
- *Assembling the model*
- *Defining your analysis steps*
- *Applying a boundary condition and a load to the model*
- *Meshing the model*
- *Creating and submitting an analysis job*
- *Viewing the results of your analysis*
- *Summary*

Understanding Abaqus/CAE modules

Abaqus/CAE is divided into modules, where each module defines an aspect of the modeling process; for example, defining the geometry, defining material properties, and generating a mesh.
As you move from module to module, you build the model from which Abaqus/CAE generates an input file that you submit to Abaqus/Standard or Abaqus/Explicit for analysis. For example, you use the Property module to define material and section properties and the Step module to choose an analysis procedure. The Abaqus/CAE postprocessor is called the Visualization module and is also licensed as a separate product called Abaqus/Viewer.

You enter a module by selecting it from the **Module** list in the context bar, as shown in *Figure 1*.



Figure 1: Selecting a module.

For the cantilever beam tutorial, you will enter the following Abaqus/CAE modules and perform the following tasks:

Part

Sketch a two-dimensional profile and create a part representing the cantilever beam.

Property

Define the material properties and other section properties of the beam.

Assembly

Assemble the model and create sets.

Step

Configure the analysis procedure and output requests.

Load

Apply loads and boundary conditions to the beam.

Mesh

Mesh the beam.

Optimization

Create an optimization task (not discussed here).

Job

Create a job and submit it for analysis.

Visualization

View the results of the analysis.

Although the **Module** list in the context bar lists the modules in a logical sequence, you can move back and forth between modules at will. However, certain obvious restrictions apply; for example, you cannot assign section properties to geometry that has not yet been created.

A completed model contains everything that Abaqus/CAE needs to generate an input file and start the analysis. Abaqus/CAE uses a model database to store your models. When you start Abaqus/CAE, the **Start Session** dialog box allows you to create a new, empty model database in memory. After you start Abaqus/CAE, you can save your model database to a disk by selecting **File->Save** from the main menu bar; to retrieve it from a disk, select **File->Open**.

For a complete listing of which module generates a particular keyword, see *Abaqus keyword browser table*.

Understanding the Model Tree

The Model Tree provides a visual description of the hierarchy of items in a model. *Figure 1* shows a typical Model Tree.

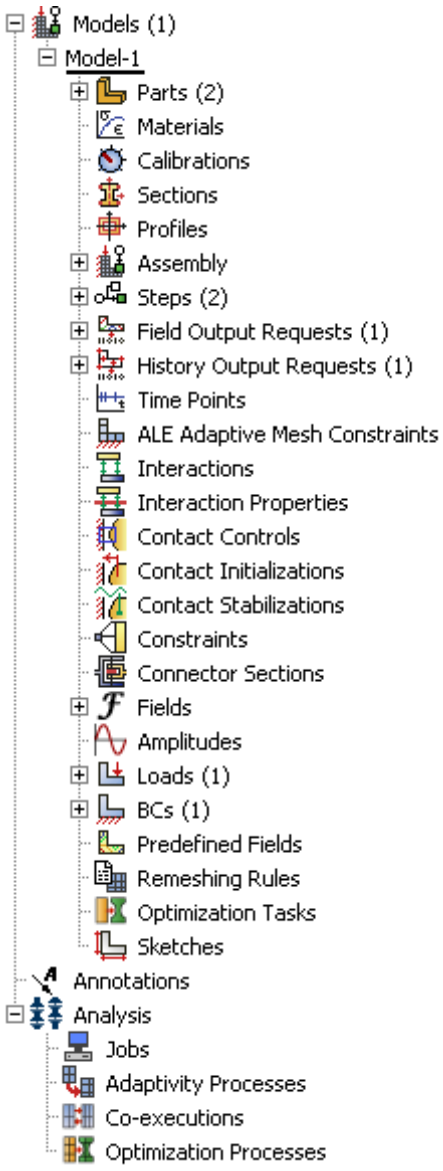


Figure 1: Model Tree.

Items in the Model Tree are represented by small icons; for example, the **Steps** icon, . In addition, parentheses next to an item indicate that the item is a container, and the number in the parentheses indicates the number of items in the container. You can click on the "+" and "−" signs in the Model Tree to expand and collapse a container. The right and left arrow keys perform the same operation.

The arrangement of the containers and items in the Model Tree reflects the order in which you are expected to create your model. As noted earlier, a similar logic governs the order of modules in the module menu—you create parts before

you create the assembly, and you create steps before you create loads. This arrangement is fixed—you cannot move items in the Model Tree.

The Model Tree provides most of the functionality of the main menu bar and the module managers. For example, if you double-click on the **Parts** container, you can create a new part (the equivalent of selecting **Part->Create** from the main menu bar).

The instructions for the examples that follow will focus on using the Model Tree to access the functionality of Abaqus/CAE. Menu bar actions will be considered only when necessary (e.g., when creating a finite element mesh or postprocessing results).

Creating a part

You can create parts that are native to Abaqus/CAE, or you can import parts created by other applications either as a geometric representation or as a finite element mesh.

You will start the cantilever beam tutorial by creating a three-dimensional, deformable solid body. You do this by sketching the two-dimensional profile of the beam (a rectangle) and extruding it. Abaqus/CAE automatically enters the Sketcher when you create a part.

Abaqus/CAE often displays a short message in the prompt area indicating what it expects you to do next, as shown in *Figure 1*.

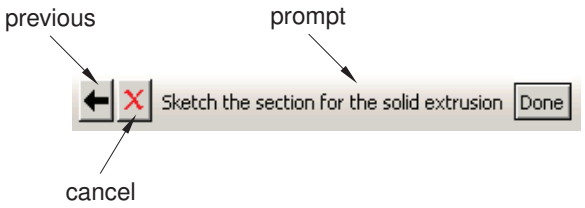


Figure 1: Messages and instructions are displayed in the prompt area.

Click the **Cancel** button to cancel the current task. Click the **Previous** button to cancel the current step in the task and return to the previous step.

1. If you did not already start Abaqus/CAE, type `abaqus cae`. Resize your windows so that you can follow the tutorial and see the Abaqus/CAE main window.
2. From the **Create Model Database** options in the **Start Session** dialog box that appears, select **With Standard/Explicit Model**. If you are already in an Abaqus/CAE session, select **File->New Model Database->With Standard/Explicit Model** from the main menu bar.

   Abaqus/CAE enters the Part module. The Model Tree appears in the left side of the main window. Between the Model Tree and the canvas is the Part module toolbox. A toolbox contains a set of icons that allow expert users to bypass the menus in the main menu bar. For many tools, as you select an item from the main menu bar or the Model Tree, the corresponding tool is highlighted in the module toolbox so you can learn its location.
3. In the Model Tree, double-click the **Parts** container to create a new part.

   The **Create Part** dialog box appears. Abaqus/CAE also displays text in the prompt area near the bottom of the window to guide you through the procedure.

   You use the **Create Part** dialog box to name the part; to choose its modeling space, type, and base feature; and to set the approximate size. You can edit and rename a part after you create it; you can also change its modeling space and type but not its base feature.
4. Name the part `Beam`. Accept the default settings of a three-dimensional, deformable body and a solid, extruded base feature. In the **Approximate size** text field, type `300`.
5. Click **Continue** to exit the **Create Part** dialog box.

   Abaqus/CAE automatically enters the Sketcher. The Sketcher toolbox appears in the left side of the main window, and the Sketcher grid appears in the viewport. The Sketcher contains a set of basic tools that allow you to sketch the two-dimensional profile of your part. Abaqus/CAE enters the Sketcher whenever you create or edit a part. To finish using a Sketcher tool, click mouse button 2 in the viewport or select a new tool.

   Tip: Like all tools in Abaqus/CAE, if you simply position the cursor over a tool in the Sketcher toolbox for a short time, a small window appears that gives a brief description of the tool.

The following aspects of the Sketcher help you sketch the desired geometry:

- The Sketcher grid helps you position the cursor and align objects in the viewport.
- Dashed lines indicate the *X*- and *Y*-axes of the sketch and intersect at the origin of the sketch.
- A triad in the lower-left corner of the viewport indicates the relationship between the sketch plane and the orientation of the part.
- When you select a sketching tool, Abaqus/CAE displays the *X*- and *Y*-coordinates of the cursor in the upper-left corner of the viewport.

6. To sketch the profile of the cantilever beam, you need to select the rectangle drawing tool .

   The rectangle drawing tool appears in the Sketcher toolbox with a white background indicating that you selected it. Abaqus/CAE displays prompts in the prompt area to guide you through the procedure.

7. In the viewport, sketch the rectangle using the following steps:
   a. You will first sketch a rough approximation of the beam and then use constraints and dimensions to refine the sketch. Select any two points as the opposite corners of the rectangle.
   b. Click mouse button 2 anywhere in the viewport to exit the rectangle tool.

      Note:

      If you are a Windows user with a 2-button mouse, press both mouse buttons simultaneously whenever you are asked to press mouse button 2.

   c. The Sketcher automatically adds constraints to the sketch (in this case the four corners of the rectangle are assigned perpendicular constraints and one edge is designated as horizontal).
   d. Use the dimension tool to dimension the top and left edges of the rectangle. The top edge should have a horizontal dimension of `200` mm, and the left edge should have a vertical dimension of `20` mm. When dimensioning each edge, simply select the line, click mouse button 1 to position the dimension text, and then enter the new dimension in the prompt area.
   e. The final sketch is shown in *Figure 2*.

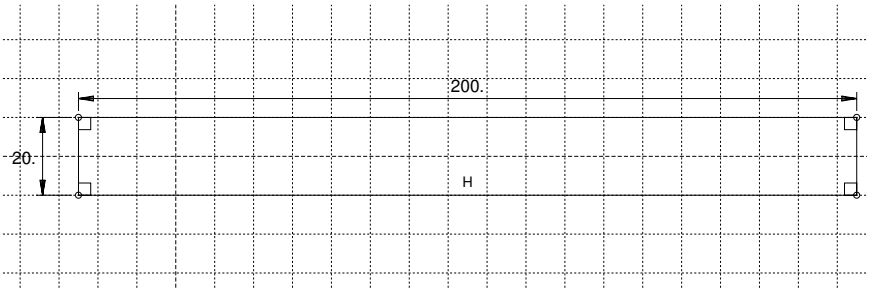


Figure 2: Sketch of the rectangle.

8. If you make a mistake while using the Sketcher, you can delete lines in your sketch, as explained in the following procedure:

   a. From the Sketcher toolbox, click the **Delete** tool, .
   b. From the sketch, click a line to select it.

      Abaqus/CAE highlights the selected line in red.

   c. Click mouse button 2 in the viewport to delete the selected line.

d. Repeat steps b and c as often as necessary.

e. Click mouse button 2 in the viewport to finish using the **Delete** tool.

Note:

You can also use the **Undo** tool and the **Redo** tool to undo and redo your previous operations.

9. From the prompt area (near the bottom of the main window), click **Done** to exit the Sketcher.

Note:

If you do not see the **Done** button in the prompt area, continue to click mouse button 2 in the viewport until it appears.

10. Because you are creating an extruded part, Abaqus/CAE displays the **Edit Base Extrusion** dialog box for you to select the depth. Optional parameters to modify the extrusion shape are also available. In the **Depth** field, erase the default value and type a value of `25.0`. Click **OK** to accept this value.

Abaqus/CAE displays an isometric view of the new part, as shown in *Figure 3*.

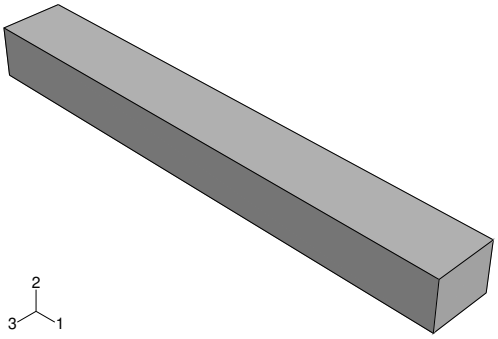


Figure 3: Isometric view of the beam.

To help you orient the cantilever beam during the modeling process, Abaqus/CAE displays a triad in the lower-left corner indicating the orientation of the global coordinate system.

11. Before you continue the tutorial, save your model in a model database file.

a. From the main menu bar, select **File->Save**. The **Save Model Database As** dialog box appears.

b. Type a name for the new model database in the **File Name** field, and click **OK**. You do not need to include the file extension; Abaqus/CAE automatically appends `.cae` to the file name.

Abaqus/CAE stores the model database in a new file and returns to the Part module. The title bar of the Abaqus/CAE window displays the path and name of the model database. You should always save your model database at regular intervals (for example, each time you switch modules).

Creating a material

For the cantilever beam tutorial you will create a single linear elastic material with Young's modulus of 209×10^3 MPa and Poisson's ratio of 0.3.

1. In the Model Tree, double-click the **Materials** container to create a new material.

   Abaqus/CAE switches to the Property module, and the **Edit Material** dialog box appears.

2. Name the material `Steel`. Use the menu bar under the browser area of the material editor to reveal menus containing all the available material options. Some of the menu items contain submenus; for example, *Figure 1* shows the options available under the **Mechanical->Elasticity** menu item. When you select a material option, the appropriate data entry form appears below the menu.

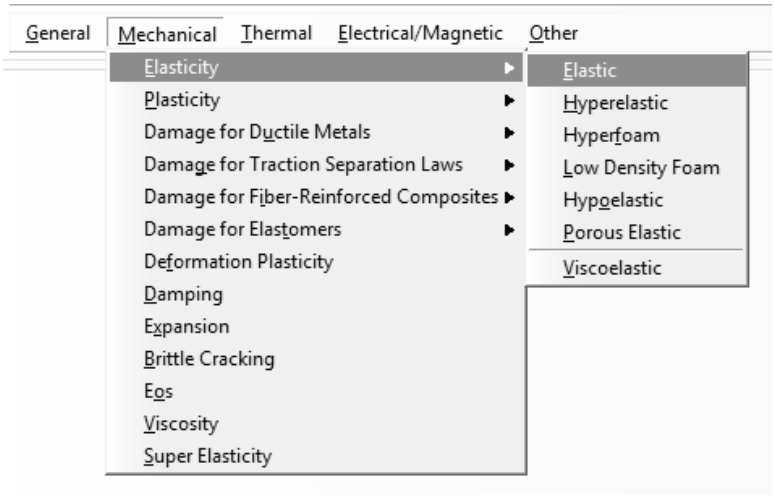


Figure 1: Submenus available under the Mechanical menu.

3. From the material editor's menu bar, select **Mechanical->Elasticity->Elastic**.

   Abaqus/CAE displays the **Elastic** data form.

4. Type a value of `209.E3` MPa for Young's modulus and a value of `0.3` for Poisson's ratio in the respective fields, as shown in *Figure 2*. Use **[Tab]** to move between cells.

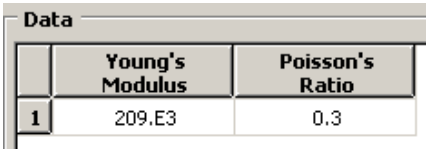


Figure 2: Entering data values for the elastic material properties.

5. Click **OK** to exit the material editor.

Defining and assigning section properties

You define the properties of a part through sections.
After you create the section, you can use one of the following two methods to assign the section to the part in the current viewport:

- You can simply select the region from the part and assign the section to the selected region.
- You can use the Set toolset to create a homogeneous set containing the region and assign the section to the set.

For the cantilever beam tutorial you will create a single homogeneous solid section that you will assign to the beam by selecting the beam from the viewport. The solid section will contain a reference to the material `Steel` that you just created.

In this section:

- *Defining a homogeneous solid section*
- *Assigning the section to the cantilever beam*

Defining a homogeneous solid section

A homogeneous solid section is the simplest section type that you can define; it includes only a material reference and an optional plane stress/plane strain thickness definition.

1. In the Model Tree, double-click the **Sections** container to create a section.

   The **Create Section** dialog box appears.

2. In the **Create Section** dialog box:
   a. Name the section `BeamSection`.
   b. In the **Category** list, accept **Solid** as the default category selection.
   c. In the **Type** list, accept **Homogeneous** as the default type selection.
   d. Click **Continue**.

      The **Edit Section** dialog box appears.

3. In the dialog box:
   a. Accept the default selection of `Steel` for the **Material** associated with the section.
   b. Click **OK**.

Assigning the section to the cantilever beam

The section `BeamSection` must be assigned to the part.

1. In the Model Tree, expand the branch for the part named **Beam** by clicking the "+" symbol to expand the **Parts** container and then clicking the "+" symbol to expand the **Beam** item.
2. Double-click **Section Assignments** in the list of part attributes that appears.
   Abaqus/CAE displays prompts in the prompt area to guide you through the procedure.
3. Click anywhere on the beam to select the region to which the section will be applied.
   Abaqus/CAE highlights the entire beam.
4. Click mouse button 2 in the viewport or click **Done** in the prompt area to accept the selected geometry.
   The **Edit Section Assignment** dialog box appears containing a list of existing sections.
5. Accept the default selection of `BeamSection` as the section, and click **OK**.
   Abaqus/CAE assigns the solid section to the beam, colors the entire beam aqua to indicate that the region has a section assignment, and closes the **Edit Section Assignment** dialog box.



Note:

When you assign a section to a region of a part, the region takes on the material properties associated with the section.

Assembling the model

Each part that you create is oriented in its own coordinate system and is independent of the other parts in the model. Although a model may contain many parts, it contains only one assembly. You define the geometry of the assembly by creating instances of a part and then positioning the instances relative to each other in a global coordinate system. An instance may be classified as independent or dependent. Independent part instances are meshed individually while the mesh of a dependent part instance is associated the mesh of the original part. This issue is discussed further in *Working with part instances*.

For the cantilever beam tutorial you will create a single instance of your cantilever beam. Abaqus/CAE positions the instance so that the origin of the sketch that defined the rectangular profile of the beam overlays the origin of the assembly's default coordinate system.

1. In the Model Tree, expand the **Assembly** container. Then double-click **Instances** in the list that appears.

   Abaqus/CAE switches to the Assembly module, and the **Create Instance** dialog box appears.

2. In the dialog box, select `Beam` and click **OK**.

   Abaqus/CAE creates an instance of the cantilever beam and displays it using an isometric orientation. In this example the single instance of the beam defines the assembly. A second triad in the viewport indicates the origin and orientation of the global coordinate system.

3. In the View Manipulation toolbar, click the rotate view manipulation tool, .

   When you move the mouse back into the viewport, a circle appears.

4. Drag the mouse in the viewport to rotate the model and examine it from all sides. You can also pick a center of rotation by clicking **Select** in the prompt area; your selected center of rotation is retained for the current object and viewport. Click **Use Default** to return to the default (center of viewport) rotation method.

   Click mouse button 2 to exit rotate mode.

5. Several other tools (pan , magnify , zoom , and auto-fit ) are also available in the View Manipulation toolbar to help you examine your model. Experiment with each of these tools until you are comfortable with them. Use the context-sensitive help system  to obtain any additional information you require about these tools.

   Direct view manipulation is available using the 3D compass. The compass allows you to pan or rotate your model by clicking and dragging on it. For example:

   - Click and drag one of the straight axes of the 3D compass to pan along an axis.
   - Click and drag any of the quarter-circular faces on the 3D compass to pan along a plane.
   - Click and drag one of the three arcs along the perimeter of the 3D compass to rotate the model about the axis that is perpendicular to the plane containing the arc.
   - Click and drag the free rotation handle (the point at the top of the 3D compass) to rotate the model freely about its pivot point.
   - Click the label for any of the axes on the 3D compass to select a predefined view (the selected axis is perpendicular to the plane of the viewport).
   - Double-click anywhere on the 3D compass to specify a view.

   The 3D compass is discussed further in *The 3D compass*.

Defining your analysis steps

Now that you have created your part, you can define your analysis steps.
For the cantilever beam tutorial the analysis will consist of two steps:

- An initial step, in which you will apply a boundary condition that constrains one end of the cantilever beam.
- A general, static analysis step, in which you will apply a pressure load to the top face of the beam.

Abaqus/CAE generates the initial step automatically, but you must create the analysis step yourself. You may also request output for any steps in the analysis.

In this section:

- *Creating an analysis step*
- *Requesting data output*

Creating an analysis step

Create a general, static step that follows the initial step of the analysis.

1. In the Model Tree, double-click the **Steps** container to create a step.

   Abaqus/CAE switches to the Step module. The **Create Step** dialog box appears with a list of all the general procedures and a default step name of `Step-1`. General procedures are those that can be used to analyze linear or nonlinear response.

2. Name the step `BeamLoad`.
3. From the list of available general procedures in the **Create Step** dialog box, select **Static, General** if it is not already selected and click **Continue**.

   The **Edit Step** dialog box appears with the default settings for a general, static step.

4. The **Basic** tab is selected by default. In the **Description** field, type `Load the top of the beam`.
5. Click the **Incrementation** tab, and accept the default time incrementation settings.
6. Click the **Other** tab to see its contents; you can accept the default values provided for the step.
7. Click **OK** to create the step and to exit the **Edit Step** dialog box.

Requesting data output

When you submit your job for analysis, Abaqus/Standard or Abaqus/Explicit writes the results of the analysis to the output database. For each step you create, you can use the **Field Output Requests Manager** and the **History Output Requests Manager** to do the following:

- Select the region of the model for which Abaqus will generate data.
- Select the variables that Abaqus will write to the output database.
- Select the section points of beams or shells for which Abaqus will generate data.
- Change the frequency at which Abaqus will write data to the output database.

When you create a step, Abaqus/CAE generates a default output request for the step. See *Which variables are in the output database?* for more information on field and history output.

For the cantilever beam tutorial, you will simply examine the output requests and accept the default configuration.

1. In the Model Tree, click mouse button 3 on the **Field Output Requests** container and select **Manager** from the menu that appears.

   Abaqus/CAE displays the **Field Output Requests Manager**. This manager displays an alphabetical list of existing output requests along the left side of the dialog box. The names of all the steps in the analysis appear along the top of the dialog box in the order of execution. The table formed by these two lists displays the status of each output request in each step.

2. Review the default output request that Abaqus/CAE generates for the **Static, General** step you created and named `BeamLoad`.

   Click the cell in the table labeled **Created**; that cell becomes highlighted, and the following information related to the cell appears in the legend at the bottom of the manager:

   - The type of analysis procedure carried out in the step in that column.
   - The list of output request variables.
   - The output request status.

3. On the right side of the **Field Output Requests Manager**, click **Edit** to view more detailed information about the output request.

   The field output editor appears. In the **Output Variables** region of the dialog box, a text box lists all the variables that will be output. If you change an output request, you can always return to the default settings by clicking **Preselected defaults** above the text box.

4. Click the arrows next to each output variable category to see exactly which variables will be output. The check boxes next to each category title allow you to see at a glance whether all variables in that category will be output. A black check mark on a white background indicates that all variables will be output, while a dark gray check mark on a light gray background indicates that only some variables will be output.

   Based on the selections shown at the bottom of the dialog box, data will be generated at every default section point in the model and will be written to the output database after every increment during the analysis.

5. Click **Cancel** to close the field output editor, since you do not wish to make any changes to the default choice.
6. Click **Dismiss** to close the **Field Output Requests Manager**.



Note:

What is the difference between the **Dismiss** and **Cancel** buttons? Dismiss buttons appear in dialog boxes that contain data that you cannot modify. For example, the **Field Output Requests Manager** allows you to view output requests, but you must use the field output editor to modify those requests. Clicking the **Dismiss** button simply closes the **Field Output Requests Manager**. Conversely, **Cancel** buttons appear in dialog boxes that allow you to make changes. Clicking **Cancel** closes the dialog box without saving your changes.

7. Review the history output requests in a similar manner by clicking mouse button 3 on the **History Output Requests** container in the Model Tree and then opening the history output editor.

Applying a boundary condition and a load to the model

Prescribed conditions, such as loads and boundary conditions, are step-dependent, which means that you must specify the step or steps in which they become active.
Now that you have defined the steps in the analysis, you can define the following prescribed conditions:

- A boundary condition that constrains one end of the cantilever beam in the *X*-, *Y*-, and *Z*-directions; the boundary condition is applied during the initial step.
- A load that you apply to the top face of the beam; the load is applied during the general analysis step.

In this section:

- *Applying a boundary condition to one end of the cantilever beam*
- *Applying a load to the top of the cantilever beam*

Applying a boundary condition to one end of the cantilever beam

Create a boundary condition that constrains the cantilever beam to be built-in at one end of the beam.

1. In the Model Tree, double-click the **BCs** container.

   Abaqus/CAE switches to the Load module, and the **Create Boundary Condition** dialog box appears.

2. In the **Create Boundary Condition** dialog box:
   a. Name the boundary condition `Fixed`.
   b. From the list of steps, select `Initial` as the step in which the boundary condition will be activated.
   c. In the **Category** list, accept **Mechanical** as the default category selection.
   d. In the **Types for Selected Step** list, accept **Symmetry/Antisymmetry/Encastre** as the default type selection, and click **Continue**.

      Abaqus/CAE displays prompts in the prompt area to guide you through the procedure.

3. You will fix the face at the left end of the cantilever beam; the desired face is shown in *Figure 1*.

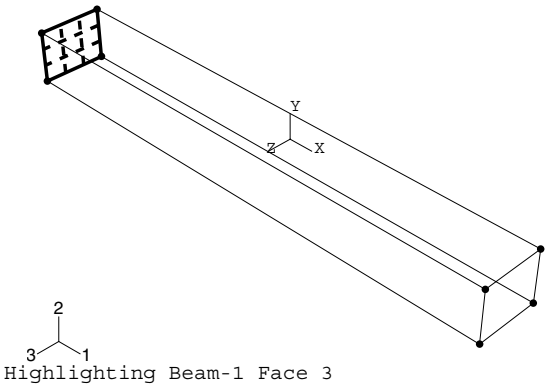


Figure 1: Selecting the region on which to apply a boundary condition.

   Rotate the view in order to select the face. Click **OK** to confirm your choice.

4. Click mouse button 2 in the viewport or click **Done** in the prompt area to indicate that you have finished selecting.

   The **Edit Boundary Condition** dialog box appears.

5. In the dialog box:
   a. Toggle on **ENCASTRE**.
   b. Click **OK** to create the boundary condition and to close the dialog box.

      Abaqus/CAE displays arrows at each corner and midpoint on the selected face to indicate the constrained degrees of freedom. Single-headed arrows represent a constraint that is applied to a translational degree of freedom. Double-headed arrows represent a constraint that is applied to a rotational degree of freedom. An **ENCASTRE** boundary condition constrains all available degrees of freedom.

6. In the Model Tree, click mouse button 3 on the **BCs** container and select **Manager** from the menu that appears.

   Abaqus/CAE displays the **Boundary Condition Manager**. The manager indicates that the boundary condition is `Created` (activated) in the initial step and is `Propagated` (continues to be active) in the general analysis step `BeamLoad`.

7. Click **Dismiss** to close the **Boundary Condition Manager**.

Applying a load to the top of the cantilever beam

Now that you have fixed one end of the cantilever beam, you can apply a distributed load to the top face of the beam. The load is applied during the general, static step you created earlier.

1. In the Model Tree, double-click the **Loads** container.

   The **Create Load** dialog box appears.

2. In the **Create Load** dialog box:
   a. Name the load `Pressure`.
   b. From the list of steps, select `BeamLoad` as the step in which the load will be applied.
   c. In the **Category** list, accept **Mechanical** as the default category selection.
   d. In the **Types for Selected Step** list, select **Pressure**, and click **Continue**.

      Abaqus/CAE displays prompts in the prompt area to guide you through the procedure.

3. In the viewport, select the top face of the beam as the surface to which the load will be applied. The desired face is shown by the gridded face in *Figure 1*.

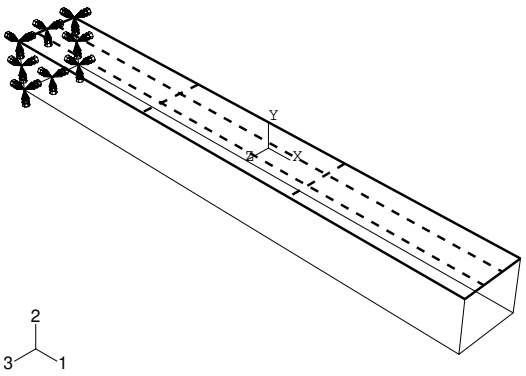


Figure 1: Selecting the region on which to apply a pressure load.

4. Click mouse button 2 in the viewport or click **Done** in the prompt area to indicate that you have finished selecting regions.

   The **Edit Load** dialog box appears.

5. In the dialog box:
   a. Enter a magnitude of `0.5` MPa for the load.
   b. Accept the default **Distribution** selection—Abaqus will apply the load uniformly over the face.
   c. Accept the default **Amplitude** selection—Abaqus will ramp up the load during the step.
   d. Click **OK** to create the load and to close the dialog box.

      Abaqus/CAE displays downward-pointing arrows along the top face of the beam to indicate the load applied in the negative 2-direction.

6. Examine the **Load Manager** and note that the new load is "Created" (activated) in the general analysis step `BeamLoad`.
7. Click **Dismiss** to close the **Load Manager**.

Meshing the model

You will now generate the finite element mesh.
You can choose the meshing technique that Abaqus/CAE will use to create the mesh, the element shape, and the element type. Abaqus/CAE uses a number of different meshing techniques. The default meshing technique assigned to the model is indicated by the color of the model when you enter the Mesh module; if Abaqus/CAE displays the model in orange, it cannot be meshed without assistance from you.

In this section:

- *Assigning mesh controls*
- *Assigning an Abaqus element type*
- *Creating the mesh*

Assigning mesh controls

In this section you will use the **Mesh Controls** dialog box to examine the technique that Abaqus/CAE will use to mesh the model and the shape of the elements that Abaqus/CAE will generate.

1. In the Model Tree, expand the **Beam** item underneath the **Parts** container and double-click **Mesh** in the list that appears.

   Abaqus/CAE switches to the Mesh module. The Mesh module functionality is available only through menu bar items or toolbox icons.

2. From the main menu bar, select **Mesh->Controls**.

   The **Mesh Controls** dialog box appears. Abaqus/CAE colors the regions of your model to indicate which technique it will use to mesh that region. Abaqus/CAE will use structured meshing to mesh your cantilever beam and displays the beam in green.

3. In the dialog box, accept **Hex** as the default **Element Shape** selection.
4. Accept **Structured** as the default **Technique** selection.
5. Click **OK** to assign the mesh controls and to close the dialog box.

   Abaqus/CAE will use the structured meshing technique to create a mesh of hexahedral-shaped elements.

Assigning an Abaqus element type

In this section you will use the **Element Type** dialog box to assign a particular Abaqus element type to the model. Although you will assign the element type now, you could also wait until after the mesh has been created.

1. From the main menu bar, select **Mesh->Element Type**.

   The **Element Type** dialog box appears.

2. In the dialog box, accept the following default selections that control the elements that are available for selection:

   - **Standard** is the default **Element Library** selection.
   - **Linear** is the default **Geometric Order**.
   - **3D Stress** is the default **Family** of elements.

3. In the lower portion of the dialog box, examine the element shape options. A brief description of the default element selection is available at the bottom of each tabbed page.

   Since the model is a three-dimensional solid, only three-dimensional solid element types—hexahedral on the **Hex** tabbed page, triangular prism on the **Wedge** page, and tetrahedral on the **Tet** page—are shown.

4. Click the **Hex** tab, and choose **Incompatible modes** from the list of formulation options.

   A description of the element type C3D8I appears at the bottom of the dialog box. Abaqus/CAE will now associate C3D8I elements with the elements in the mesh.

5. Click **OK** to assign the element type and to close the dialog box.

Creating the mesh

Basic meshing is a two-stage operation: first you seed the edges of the part instance, and then you mesh the part instance. You select the number of seeds based on the desired element size or on the number of elements that you want along an edge, and Abaqus/CAE places the nodes of the mesh at the seeds whenever possible. For the cantilever beam tutorial the default seeding will generate a mesh with square hexahedral elements.

1. From the main menu bar, select **Seed->Part** to seed the part instance.

   The **Global Seeds** dialog box appears. The dialog box displays the default element size that Abaqus/CAE will use to seed the part instance. This default element size is based on the size of the part instance.

2. In the dialog box, enter an approximate global size of `10.0` and click **OK**.
3. Click **Done** in the prompt area to indicate that you have finished the seed definition.

   Abaqus/CAE applies the seeds to the part instance, as shown in *Figure 1*. You can gain more control of the resulting mesh by seeding each edge of the part instance individually.

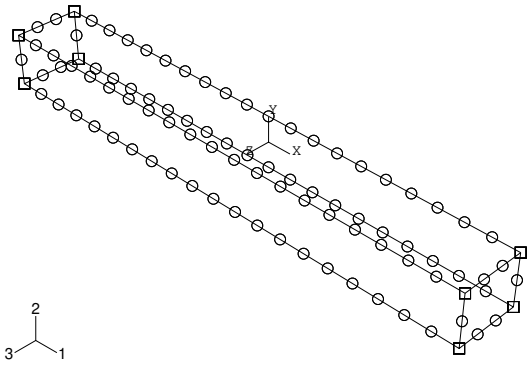


Figure 1: Seeding the mesh.

4. From the main menu bar, select **Mesh->Part** to mesh the part instance.
5. In the prompt area, click **Yes** to confirm that you want to mesh the part instance.

   Abaqus/CAE meshes the part instance and displays the resulting mesh, as shown in *Figure 2*.

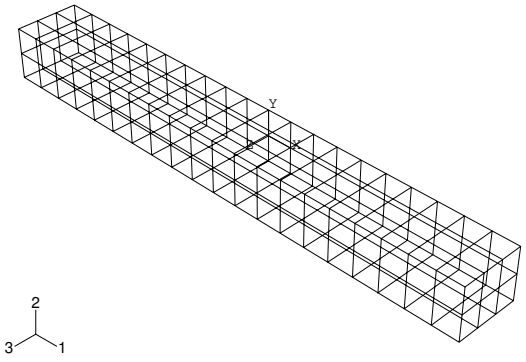


Figure 2: Meshing the part.

Creating and submitting an analysis job

Now that you have configured your analysis, you will create a job that is associated with your model and to submit the job for analysis.

1. In the Model Tree, double-click the **Jobs** container to create a job.

   Abaqus/CAE switches to the Job module, and the **Create Job** dialog box appears with a list of the models in the model database.

2. Name the job `Deform`.
3. Click **Continue** to create the job.

   The **Edit Job** dialog box appears.

4. In the **Description** field, type `Cantilever beam tutorial`.
5. Click the tabs to review the default settings in the job editor. Click **OK** to accept all the default job settings and to close the dialog box.
6. In the Model Tree, expand the **Jobs** container; click mouse button 3 on the job named **Deform**, and select **Submit** from the menu that appears to submit your job for analysis.

   After you submit your job, information appears next to the job name indicating the job's status. The status of the cantilever beam tutorial shows one of the following:

   - **Submitted** while the analysis input file is being generated.
   - **Running** while Abaqus analyzes the model.
   - **Completed** when the analysis is complete, and the output has been written to the output database.
   - **Aborted** if Abaqus/CAE finds a problem with the input file or the analysis ends abnormally. In addition, Abaqus/CAE reports the problem in the message area.

7. When the job completes successfully, you are ready to view the results of the analysis with the Visualization module. In the Model Tree, click mouse button 3 on the job named **Deform** and select **Results** to enter the Visualization module.

   Abaqus/CAE enters the Visualization module, opens the output database created by the job, and displays the undeformed model shape.

Viewing the results of your analysis

You use the Visualization module to read the output database that Abaqus/CAE generated during the analysis and to view the results of the analysis. Because you named the job `Deform` when you created the job, Abaqus/CAE names the output database `Deform.odb`.

For the tutorial you will view the undeformed and deformed shapes of the cantilever beam model and create a contour plot.

1. After you select **Results** in the Model Tree, Abaqus/CAE enters the Visualization module, opens `Deform.odb`, and displays the undeformed shape of the model, as shown in *Figure 1*.

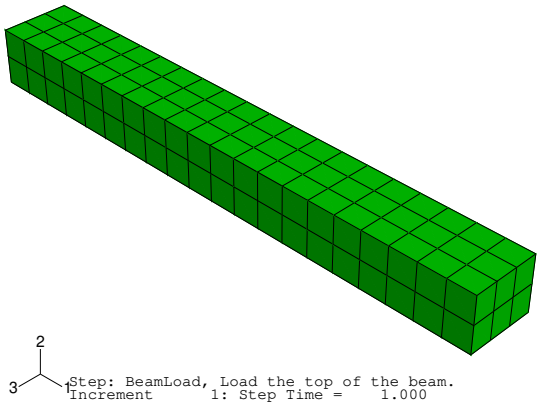


Figure 1: Undeformed shape plot of model (title block suppressed).

The title block indicates the following:

- The job description.
- The output database from which Abaqus/CAE read the data.
- The release of Abaqus/Standard or Abaqus/Explicit that was used to generate the output database.
- The date the output database was generated.

The state block indicates the following:

- The step name and the step description.
- The increment within the step.
- The step time.
- When you are viewing a deformed plot, the deformed variable and the deformation scale factor.

By default, Abaqus/CAE plots the last step and the last frame of your analysis. Buttons that allow you to control which analysis results are plotted are available in the prompt area.

2. From the main menu bar, select **Plot->Deformed Shape** to view a deformed shape plot.

3. Click the auto-fit tool  so that the entire plot is rescaled to fit in the viewport, as shown in *Figure 2*.

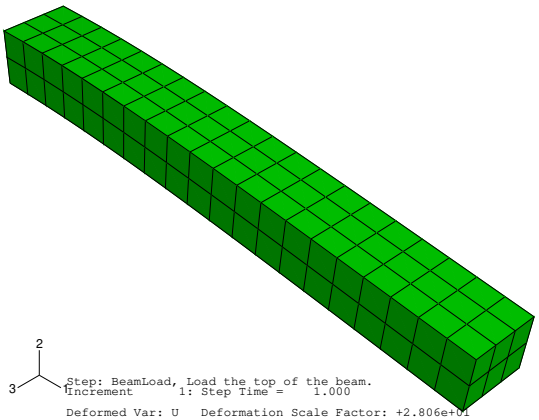


Figure 2: Deformed shape plot of model (title block suppressed).

4. From the main menu bar, select **Plot->Contours->On Deformed Shape** to view a contour plot of the von Mises stress, as shown in *Figure 3*.

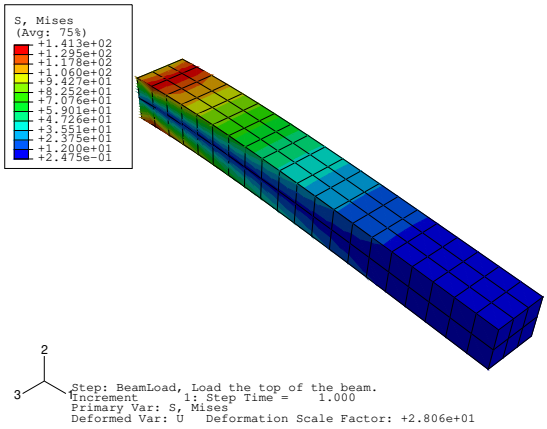


Figure 3: Contour plot of Mises stress (title block suppressed).

5. For a contour plot the default variable displayed depends on the analysis procedure; in this case, the default variable is the von Mises stress. From the main menu bar, select **Result->Field Output** to examine the variables that are available for display.

   Abaqus/CAE displays the **Field Output** dialog box; click the **Primary Variable** tab to choose which variable to display and to select the invariant or component of interest. By default, the **Mises** invariant of the **Stress components at integration points** variable is selected.

   Tip: You can also use the **Field Output** toolbar to change the displayed field output variable. For more information, see *Using the field output toolbar*.

6. Click **Cancel** to close the **Field Output** dialog box.

You have now finished this tutorial. *Using Additional Techniques to Create and Analyze a Model in Abaqus/CAE* introduces additional techniques to create and analyze a model; for example, you will create and assemble multiple part instances and define contact. *Viewing the Output from Your Analysis* covers the capabilities of the Visualization module in more detail.

Summary

This section presents a summary of the concepts.

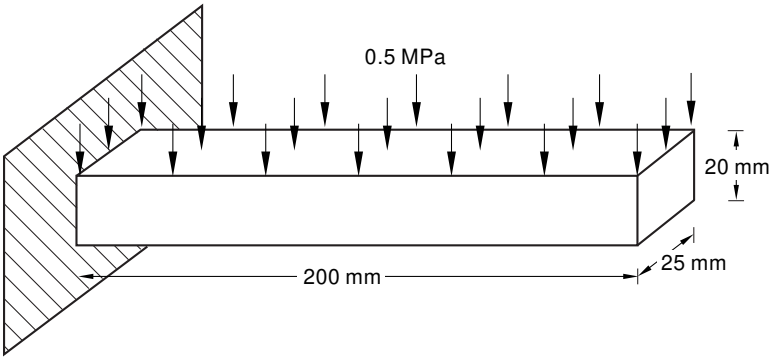


Figure 1: A loaded cantilever beam.

- When you create a part, you name it and choose its type, modeling space, base feature, and approximate size.
- Abaqus/CAE automatically enters the Sketcher when you create or edit a part. You use the Sketcher to draw the two-dimensional profiles of parts.
- Click mouse button 2 in the viewport to indicate you have finished selecting items or using a tool.
- You can create a material and define its properties and create a section and define its category and type. Since the section refers to the material, the material must be defined first.
- A model contains only one assembly. The assembly is composed of instances of parts positioned in a global coordinate system.
- Abaqus/CAE generates the initial step automatically, but you must create analysis steps. You use the step editor to define each analysis step.
- When you create a step, Abaqus/CAE generates a default output request for the step. You use the **Field Output Requests Manager** and the **History Output Requests Manager** to examine which categories of data will be output.
- You invoke the field and history output editors from the **Field Output Requests Manager** and the **History Output Requests Manager** to select the variables that Abaqus/CAE will write to the output database during the analysis, as well as the frequency at which they are written and the regions and section points from which they are written.
- Prescribed conditions, such as loads and boundary conditions, are step-dependent objects, which means that you must specify the step or steps in which they become active.
- Managers are useful for reviewing and modifying the status of prescribed conditions in each step.
- You create loads and define where the load is applied to the assembly in the Load module.
- Although you can create a mesh at any point after creating the assembly, you typically do it after configuring the rest of the model, since items such as loads, boundary conditions, and steps depend on the underlying geometry, not the mesh.
- You can assign the element type either before or after you create the mesh. The available element types depend on the geometry of your model.
- You use seeds to define the approximate position of nodes in your final mesh. You select the number of seeds based on the element size or on the number of elements that you want along an edge.
- You can use the Model Tree to submit jobs and to monitor the status of a job.

- In the Visualization module you read the output database generated by your analysis and view the results. You can select the variable to display from the data in the output database, and you can also select the increment being displayed.
- You can display the results in several modes—undeformed, deformed, and contour. You can control the appearance of the display in each mode, independent of other modes.

Using Additional Techniques to Create and Analyze a Model in Abaqus/CAE

Creating and Analyzing a Simple Model in Abaqus/CAE explains how to create and analyze a very simple model composed of only one part. In this advanced tutorial for the experienced Abaqus user you will create and analyze a more complex model. The model is more complex on two levels:

- It consists of three different parts rather than just one. This tutorial illustrates how you position instances of these parts to create the assembly and how you define contact between surfaces of the assembly.
- It includes parts that you will draw using advanced sketching techniques. You will learn how sketches, datum geometry, and partitions combine to define the features that make up individual parts. You will also learn how you can modify a part by editing a feature and how modified parts are regenerated.

As in *Creating and Analyzing a Simple Model in Abaqus/CAE*, you will apply section properties, loads, and boundary conditions to the model; you will also mesh the model, configure the analysis, and run the analysis job. At the end of the tutorial you will view your analysis results. The entire tutorial takes approximately three hours to complete.

This tutorial assumes that you are familiar with the techniques described in *Creating and Analyzing a Simple Model in Abaqus/CAE*, including the following:

- Using the view manipulation tools to rotate and zoom an object in the viewport.
- Following the prompts in the prompt area.
- Using the mouse to select menu items, toolbox items, and items within the viewport.

In this section:

- *Creating an assembly composed of a hinge held together by a pin*
- *Creating the first hinge piece*
- *Assigning section properties to the hinge part*
- *Creating and modifying a second hinge piece*
- *Creating the pin*
- *Assembling the model*
- *Defining analysis steps*
- *Creating surfaces to use in contact interactions*
- *Defining contact between regions of the model*
- *Applying boundary conditions and loads to the assembly*
- *Meshing the assembly*
- *Creating and submitting a job*
- *Viewing the results of your analysis*
- *Summary*

Creating an assembly composed of a hinge held together by a pin

During the tutorial you will create an assembly composed of a hinge held together by a pin.
The assembled part instances and the final mesh are illustrated in *Figure 1*.

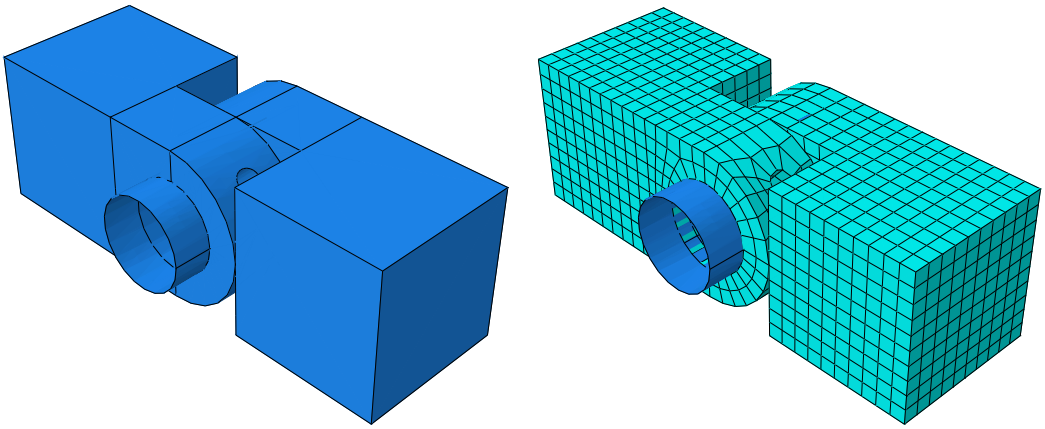


Figure 1: Model used in the hinge tutorial.

The tutorial consists of the following sections:

- *Creating the first hinge piece*
- *Assigning section properties to the hinge part*
- *Creating and modifying a second hinge piece*
- *Creating the pin*
- *Assembling the model*
- *Defining analysis steps*
- *Creating surfaces to use in contact interactions*
- *Defining contact between regions of the model*
- *Applying boundary conditions and loads to the assembly*
- *Meshing the assembly*
- *Creating and submitting a job*
- *Viewing the results of your analysis*

Creating the first hinge piece

To start the tutorial, you create the first part—half of the hinge.
Abaqus/CAE models are composed of features; you create a part by combining features. This portion of the hinge is composed of the following features:

- A cube—the base feature, since it is the first feature of the part.
- A flange that extends from the cube. The flange also includes a large-diameter hole through which the pin is inserted.
- A small lubrication hole in one corner of the flange.

In this section:

- *Creating the cube*
- *Adding the flange to the base feature*
- *Modifying a feature*
- *Creating the sketch plane*
- *Sketching the lubrication hole*

Creating the cube

To create the cube (the base feature), you create a solid, three-dimensional, extruded part and name it.

You then sketch its profile (0.04 m × 0.04 m) and extrude the profile over a specified distance (0.04 m) to produce the base feature of the first half of the hinge. The desired cube is shown in *Figure 1*.

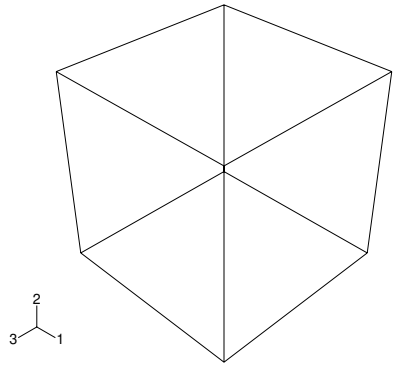


Figure 1: The base feature (a cube) is created first.

Note:

The default render style used throughout Abaqus/CAE is **Shaded**. For clarity, many of the figures in this tutorial use the wireframe or hidden line render styles. For more information, see *Choosing a render style*.

1. Start Abaqus/CAE, and create a new model database. Resize your windows so that you can follow the tutorial and see the Abaqus/CAE main window.

   Abaqus/CAE enters the Part module and displays the Model Tree in the left side of the main window.

2. In the Model Tree, double-click the **Parts** container to create a new part.

   The **Create Part** dialog box appears.

   The text in the prompt area asks you to fill out the **Create Part** dialog. Abaqus/CAE always displays prompts in the prompt area to guide you through a procedure.

3. Name the part `Hinge-hole`. Accept the following default settings:

   - A three-dimensional, deformable body
   - A solid extrusion base feature

4. In the **Approximate size** text field, type `0.2`. You will be modeling the hinge using meters for the unit of length, and its overall length is 0.14 meters; therefore, 0.2 meters is a sufficiently large approximate size for the part. Click **Continue** to create the part.

   The Sketcher starts and displays the toolbox between the canvas and the Model Tree. Abaqus/CAE uses the approximate size of the part to compute the default sheet size—0.2 meters in this example. In addition, in this example the Sketcher draws 40 gridlines on the sheet, and the distance between each gridline is 0.005 meters. (You probably see fewer than 40 gridlines because the sheet extends beyond your viewport.)

5. From the Sketcher toolbox, select the rectangle tool .
6. Sketch an arbitrary rectangle, and click mouse button 2 in the viewport to exit the rectangle tool.
7. Dimension the top and left edges so that each is `0.04` m long.



Important: To complete this tutorial successfully, it is important that you use the dimensions stated and do not deviate from the example; otherwise, you will find it difficult to assemble the model.

8. Click mouse button 2 to exit the Sketcher.



Tip: Clicking mouse button 2 in the viewport has the same effect as clicking the default button in the prompt area—**Done** in this instance.

Abaqus/CAE displays the **Edit Base Extrusion** dialog box.

9. In the dialog box, type a **Depth** of `0.04` and press **[Enter]**.

Abaqus/CAE exits the Sketcher and displays the base feature, a cube, as shown in *Figure 1*. The triad in the lower-left corner of the viewport indicates the orientation of the *X*-, *Y*-, and *Z*-axes. You can turn off this triad by selecting **Viewport->Viewport Annotation Options** from the main menu bar and toggling off the **Show triad** option. (The triad is sometimes turned off for clarity in the figures in this tutorial.)



Note:

By default, Abaqus/CAE uses the alphabetical option, *x-y-z*, for labeling the view orientation triad. In general, this guide adopts the numerical option, 1-2-3, to permit direct correspondence with degree of freedom and output labeling. For more information on labeling of axes, see *Customizing the view triad*.

Adding the flange to the base feature

You will now add a solid feature—the flange—to the base feature.

You select one face of the cube to define the sketch plane and extrude the sketched profile through half the depth of the cube. The cube and flange are shown in *Figure 1*.

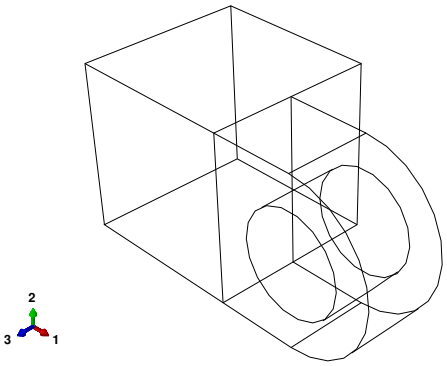


Figure 1: The flange is added to the base feature.

1. From the main menu bar, select **Shape->Solid->Extrude**.
2. Select the face at the front of the cube to define the sketching plane, as shown in *Figure 2*.

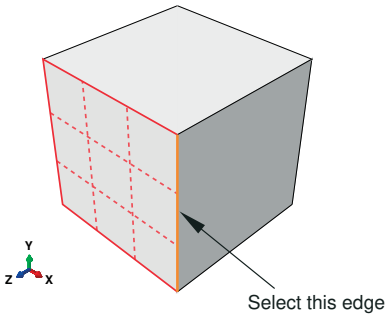


Figure 2: Select the gridded face to define the sketching plane. Select the indicated edge to position the part correctly in the Sketcher.

When you stop moving the cursor during a selection procedure, Abaqus/CAE highlights the edges of the entity that it would select at the current cursor position. This highlighting behavior is called "preselection."



Note:

Two forms of preselection are available in Abaqus/CAE: one for object selection from the viewport and the other for selection from the Sketcher. For more information, see *Highlighting objects prior to selection* and *Turning preselection on or off*, respectively.

3. Select an edge that will appear vertical and on the right side of the sketch, as shown in *Figure 2*.

   Again, Abaqus/CAE uses preselection to aid you in selecting the desired edge.

   The Sketcher starts and displays the outline of the base feature as reference geometry. Abaqus/CAE magnifies the view to fit the sketch plane; the sheet size and grid spacing are also recalculated based on the size of the sketch plane. To change the sheet size and grid spacing back to their original settings and disable their automatic recalculation for the current session, use the options tool , located in

the Sketcher toolbox. On the **General** tabbed page, toggle off **Auto** next to the sheet size text field and set the value to `0.2`; toggle off **Auto** next to the grid spacing text field and set the value to `0.005`.

Tip: To retain the original sheet size and grid spacing for all sketches in a part, you can select the options tool while sketching the base feature—the cube—and toggle off both **Auto** settings.

The sketch of the flange that you will create is illustrated in *Figure 3*. To duplicate the view in the figure, use the options tool again to double the grid spacing.

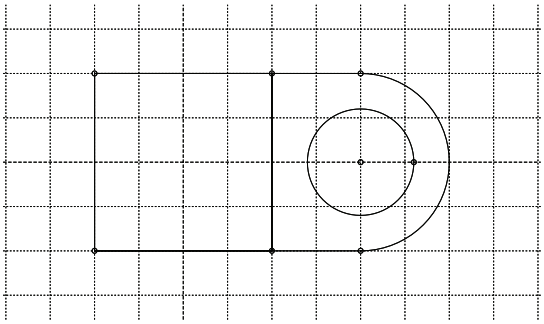


Figure 3: Use the Sketcher to create the flange profile.

4. Zoom out to view the area where you will sketch the flange:
   a. From the View Manipulation toolbar, select the magnify tool .
   b. Position the cursor near the center of the viewport.
   c. Click mouse button 1 and drag to the left until the cube occupies approximately half of the visible Sketcher space.

   Reducing the view is necessary because the flange is created beyond the edges of the selected sketch plane.
5. As before, the approximate shape of the new feature will be sketched first. From the Sketcher toolbox, select the connected lines tool .
6. Sketch the rectangular portion of the flange by drawing three lines as follows:
   a. Starting at any point to the right of the cube, connect the line to the top-right corner of the cube.
   b. Continue the next line to the bottom-right corner of the cube. This line is automatically assigned a vertical constraint.
   c. The final line extends from the bottom-right corner of the cube to any point to the right of the cube.

   Tip: If you make a mistake while sketching, use the Sketcher undo or delete tools to correct your error.
7. Click mouse button 2 in the viewport to exit the connected lines tool.
8. Refine the sketch by defining the following constraints and dimension:
   a. Use the constraints tool to constrain the top and bottom lines of the sketch so that each is horizontal.
   b. Assign an equal length constraint to these two lines (use **[Shift] + Click** to select both lines).
   c. Dimension either line so that it is `0.02` m long.

   The sketch appears as shown in *Figure 4*.

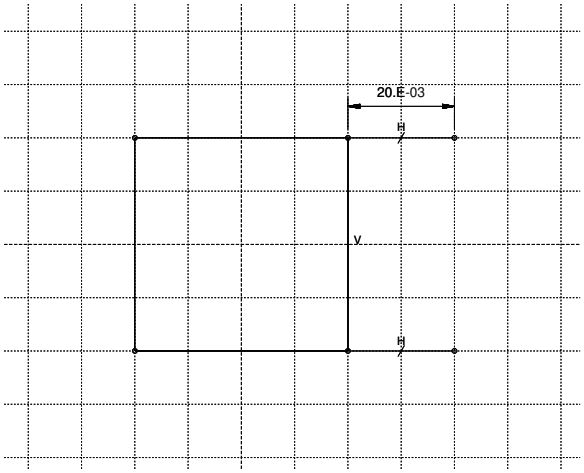


Figure 4: Draw the rectangular portion of the flange.

9. Close the profile by adding a semicircular arc using the 3-points circle tool .

a. Select the two vertices at the open end of the rectangle as the endpoints of the arc, starting with the top one. Select any point to the right of the sketch as a point that lies on the arc.

b. Define tangent constraints between the ends of the arc and the horizontal lines to refine the sketch.

10. Click mouse button 2 in the viewport to exit the 3-points circle tool.

The resulting arc is shown in *Figure 5*.

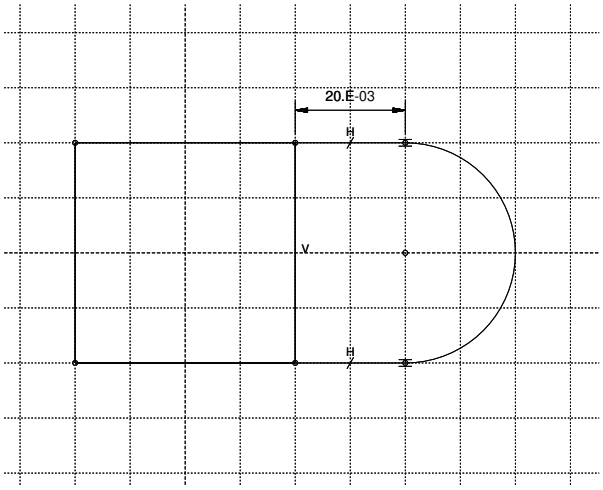


Figure 5: Add the curved portion of the flange.

11. From the Sketcher toolbox, select the center-perimeter circle tool  to sketch the flange hole.

a. Place the center of the circle to coincide approximately with the center of the arc created previously. The perimeter point should be placed to the right of the center point. Apply a concentric constraint between the two circular regions.

b. Use the dimension tool  to change the value of the radius to `0.01` m.

c. Dimension the vertical distance between the center of each circle and its perimeter point. Edit this dimension so that the distance is `0`. (If the distance is already `0`, you cannot add a vertical dimension.)

This will adjust the location of the perimeter point so that it is on the same horizontal plane as the center point.



Note:

When you mesh a part, Abaqus/CAE places nodes wherever vertices appear along an edge; therefore, the location of the vertex on the circumference of the circle influences the final mesh. Placing it on the same horizontal plane as the center point results in a high-quality mesh.

12. The final sketch is shown in *Figure 6*.

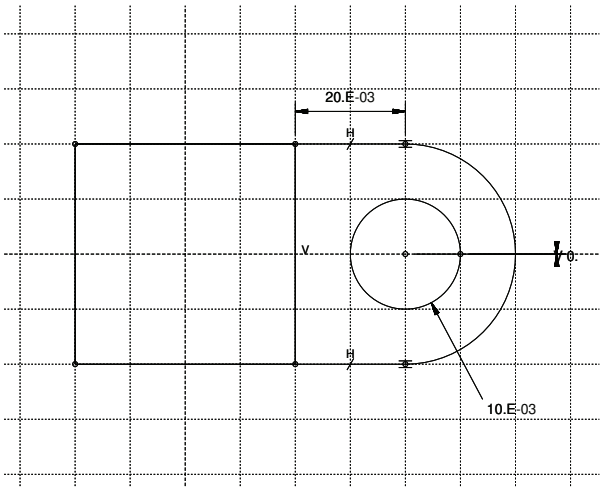


Figure 6: Final sketch.

13. Click mouse button 2 to exit the Sketcher.

Abaqus/CAE displays the part in an isometric view showing the base extrusion, your sketched profile, and an arrow indicating the extrusion direction. The default extrusion direction for a solid is always out of the solid. Abaqus/CAE also displays the **Edit Extrusion** dialog box.



Tip: Use the auto-fit view manipulation tool  to fit the sketched flange profile and the base extrusion in the viewport.

14. In the **Edit Extrusion** dialog box:

a. Accept the default **Type** selection of **Blind** to indicate that you will provide the depth of the extrusion.

b. In the **Depth** field, type an extrusion depth of `0.02`.

c. Click  to reverse the extrusion direction, as shown in *Figure 7*.

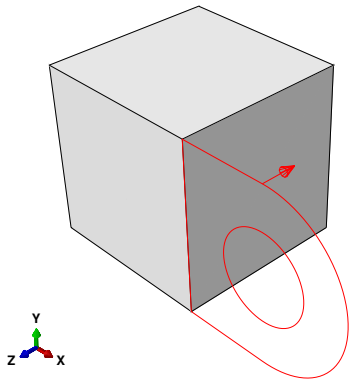


Figure 7: Completed flange sketch showing the extrusion direction.

d. Toggle on **Keep internal boundaries**. When you toggle this option on, Abaqus/CAE maintains the face that is generated between the extruded solid feature and the existing part. As a result, the extruded flange is maintained as a second cell and is not merged with the cube. (When you mesh the model at the end of the tutorial, the internal boundary allows you to mesh the flange without having to first partition the cell and flange into separate cells.)

e. Click **OK** to create the solid extrusion.

Abaqus/CAE displays the part composed of the cube and the flange. Use the auto-fit view manipulation tool  again to resize the part to fit in the viewport.

Modifying a feature

Each part is defined by a set of features, and each feature in turn is defined by a set of parameters. For example, the base feature (the cube) and the second feature (the flange) are both defined by a sketch and an extrusion depth. You modify a part by modifying the parameters that define its features. For the hinge example you will change the radius of the hole in the sketch of the flange from 0.01 m to 0.012 m.

1. In the Model Tree, expand the **Hinge-hole** item underneath the **Parts** container. Then expand the **Features** container that appears.

   A list showing each feature's **Name** appears. In this example you have created two solid extrusion features: the base feature (the cube), **Solid extrude-1**, and the flange, **Solid extrude-2**.

2. Click mouse button 3 on **Solid extrude-2** (the flange) in this list.

   Abaqus/CAE highlights the selected feature in the viewport.

3. From the menu that appears, select **Edit**.

   Abaqus/CAE displays the feature editor. For an extruded solid you can change the extrusion depth, the twist or draft (if specified when the feature was created), and the profile sketch.

4. From the feature editor, click .

   Abaqus/CAE displays the sketch of the second feature, and the feature editor disappears.

5. From the edit tools in the Sketcher toolbox, select the edit dimension value tool .
6. Select the radial dimension of the circle (`0.010`).
7. In the **Edit Dimension** dialog box, type a new radius of `0.012` and click **OK**.

   Abaqus/CAE closes the dialog box and changes the radius of the circle in the sketch only.

8. Click mouse button 2 to exit the edit dimension value tool. Click mouse button 2 again to exit the Sketcher.

   Abaqus/CAE again displays the feature editor.

9. Click **OK** to regenerate the flange with the modified radius and to exit the feature editor.

   The flange hole is enlarged to the new radius dimension.



Note:

In some circumstances regenerating a feature causes dependent features to fail. In such a case Abaqus/CAE asks if you want to save your changes and suppress the features that failed to regenerate, or if you want to revert to the unmodified feature and lose your changes.

Creating the sketch plane

You must create an appropriate datum plane on which to sketch the profile of the extruded cut.

The flange includes a small hole used for lubrication, as shown in *Figure 1*.

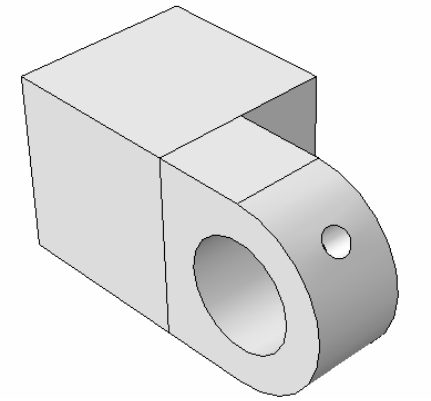


Figure 1: Isometric shaded view of the hinge with the lubrication hole.

Creating the hole in the desired location requires an appropriate datum plane on which to sketch the profile of the extruded cut, as shown in *Figure 2*.

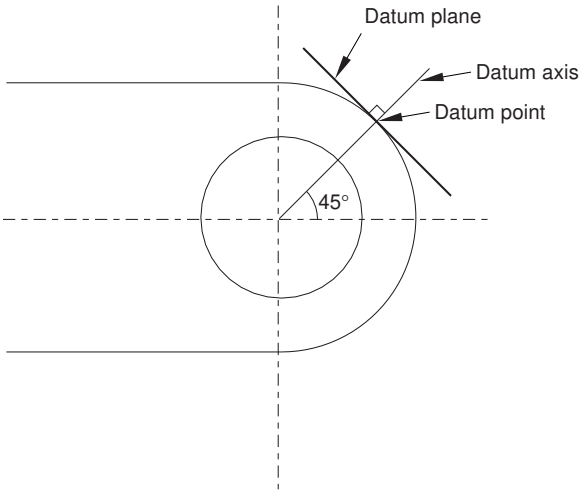


Figure 2: Two-dimensional view of the datum plane's position with respect to the hinge piece.

You sketch a circle on the datum plane, which is tangent to the flange, and Abaqus/CAE extrudes the circle normal to the datum plane and normal to the flange to create the lubrication hole.

There are three operations involved in creating the datum plane:

- Creating a datum point on the circumference of the flange.
- Creating a datum axis running between two datum points.
- Creating a datum plane through the datum point on the circumference and normal to the datum axis.

1. From the main menu bar, select **Tools->Datum**.

   Abaqus/CAE displays the **Create Datum** dialog box.

2. Create a datum point along the curved edge of the flange through which the datum plane will pass. From the **Create Datum** dialog box, choose the **Point** datum type.

3. From the list of methods, click **Enter parameter**.
4. Select the curved edge, as shown in *Figure 3*. Note the direction of the arrow indicating an increasing edge parameter from 0.0 to 1.0. You cannot change the direction of this arrow.

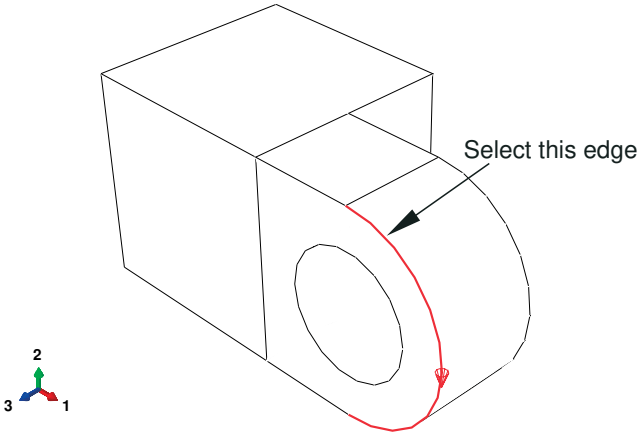


Figure 3: Create a datum point along the curved edge of the flange.

5. In the text box in the prompt area, enter a normalized edge parameter and press **[Enter]**. If the arrow direction is the same as in *Figure 3*, enter `0.25` as the normalized edge parameter; if the arrow points in the opposite direction, enter `0.75` as the normalized edge parameter.

   Abaqus/CAE creates a datum point along the selected edge.
6. Create a datum axis that will define the normal to the datum plane. From the **Create Datum** dialog box, choose the **Axis** datum type. Click the **2 points** method.

   Abaqus/CAE highlights the points that can be used to create the datum axis.
7. Select the point at the center of the hole (created when you sketched the hole's profile) and the datum point on the curved edge.

   Abaqus/CAE displays a datum axis passing through the two points, as shown in *Figure 4*.

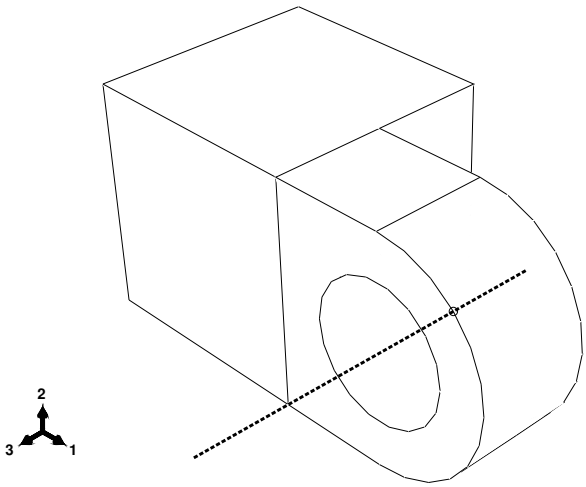


Figure 4: Create a datum axis defined by two datum points.

8. The final step is to create the datum plane normal to the datum axis. From the **Create Datum** dialog box, choose the **Plane** datum type. Click the **Point and normal** method.

9. Select the datum point on the curved edge as the point through which the datum plane will pass.
10. Select the datum axis as the edge that will be normal to the datum plane.

    Abaqus/CAE creates the datum plane, as shown in *Figure 5*.

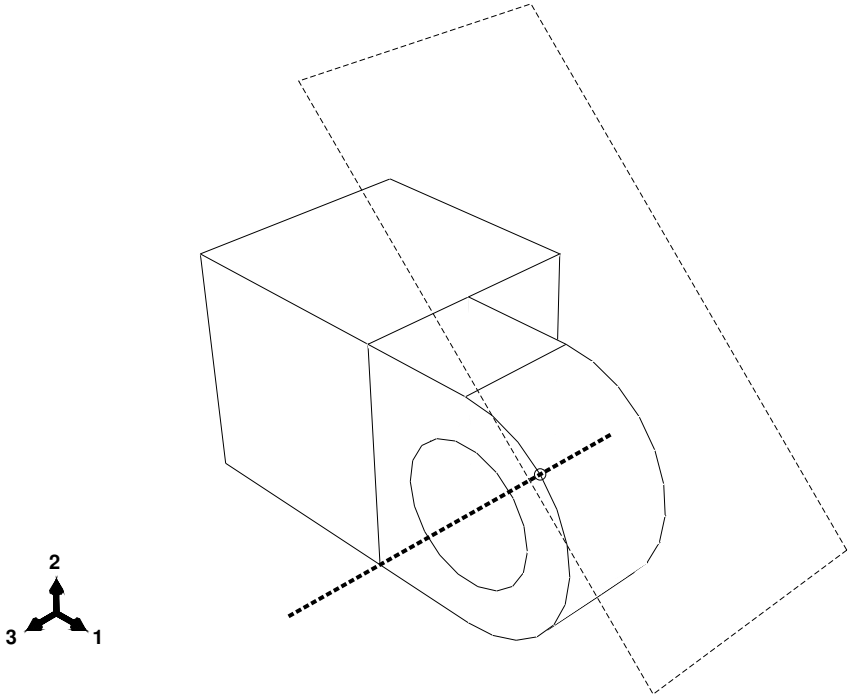


Figure 5: Create a datum plane normal to the datum axis.

Sketching the lubrication hole

The next operation creates the lubrication hole on the flange by extruding a circle from the datum plane that you just created.

First, you need to create a datum point on the flange that indicates the center of the hole, as illustrated in *Figure 1*.

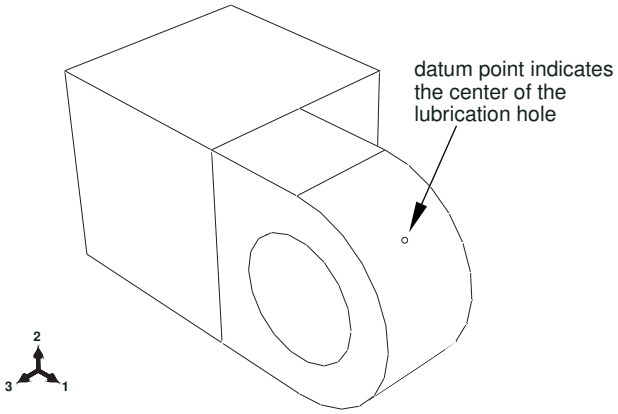


Figure 1: A datum point indicates the center of the lubrication hole.

Create the datum point at the center of the lubrication hole

1. Create a datum point along the second curved edge of the flange. From the **Create Datum** dialog box, choose the **Point** datum type.
2. From the list of methods, click **Enter parameter**.
3. Select the second curved edge of the flange, as shown in *Figure 2*.

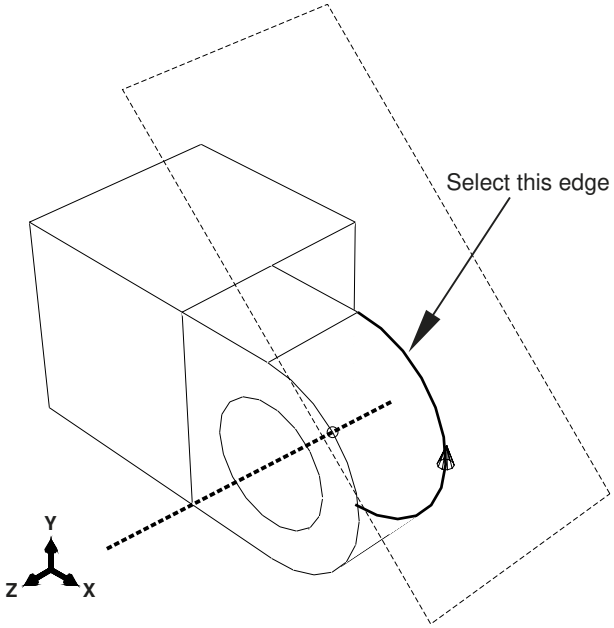


Figure 2: Select the second edge.

4. Note the direction of the arrow indicating an increasing edge parameter from 0.0 to 1.0. Enter a normalized edge parameter of `0.75` (or `0.25` if the sense of the arrow is opposite that shown in *Figure 2*), and press **[Enter]**.

   Abaqus/CAE creates a datum point along the selected edge.

5. From the list of methods in the **Create Datum** dialog box, select **Midway between 2 points**.
6. Select the datum point along the first curved edge.
7. Select the datum point along the second curved edge.

   Abaqus/CAE creates a datum point halfway across the flange.

8. Close the **Create Datum** dialog box.

   This exercise illustrates how you can use feature-based modeling to capture your design intent. The datum point is a feature that Abaqus/CAE defines to be midway between the datum points along the edges of the flange. As a result, if you change the thickness of the flange, the lubrication hole remains in the center.

Sketch the lubrication hole

1. From the main menu bar, select **Shape->Cut->Extrude**.
2. Click the boundary of the datum plane to select it as the plane on which to sketch.
3. Select the top rear edge of the cube as the edge that will appear vertical and on the right side of the sketch, as shown in *Figure 3*.

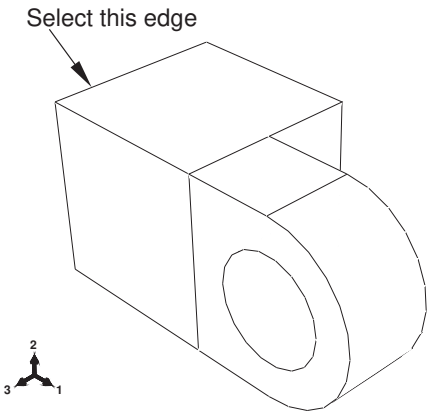


Figure 3: Select the indicated edge to position the part correctly in the Sketcher grid.

The Sketcher starts with the vertices, datums, and edges of the part projected onto the sketch plane as reference geometry.



Tip: If you are unsure of the relative orientation of the sketch plane and the part, use the view manipulation tools to rotate and pan them. Use the reset view tool  to restore the original view.

4. From the Sketcher toolbox, select the circle tool .
5. Select the datum point on the center of the flange to indicate the center of the circle.
6. Select any other point, and click mouse button 1.
7. Dimension the radius of the hole. The radius of the circle should be changed to 0.003 m.

8. Dimension the vertical distance between the circle's center and perimeter points. Set this distance to zero. As noted earlier, this will improve the quality of the mesh.
9. Exit the Sketcher.

   Abaqus/CAE displays the hinge in an isometric view showing the base part and flange, your sketched hole profile, and an arrow indicating the direction for the extruded cut. Abaqus/CAE also displays the **Edit Cut Extrusion** dialog box.
10. From the **Type** menu in the **Edit Cut Extrusion** dialog box, select **Up to Face** and click **OK**.
11. Select the cylindrical inner surface of the hole in the part to indicate the face to which to extrude, as illustrated in *Figure 4*. (Because you can select at most only one face, Abaqus/CAE does not ask you to indicate that you have finished selecting.)

    Abaqus/CAE extrudes the sketch from the datum plane to the hole in the flange.

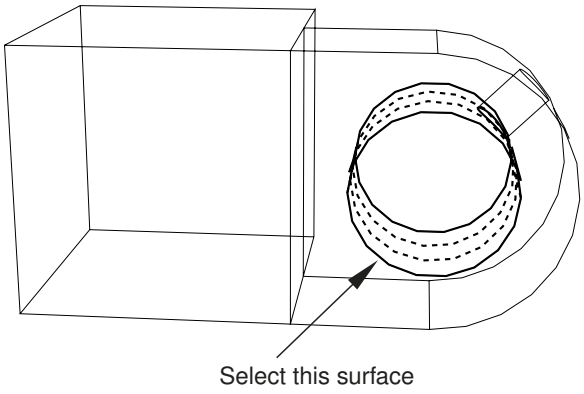


Figure 4: Select the face to which to extrude.

12. From the Render Style toolbar, select the shaded display tool  if necessary, and use the rotation tool  to see how the part and its features are oriented, as shown in *Figure 5*. (For clarity, the datum geometry has been removed from the view in *Figure 5* by selecting **View->Part Display Options->Datum**.)

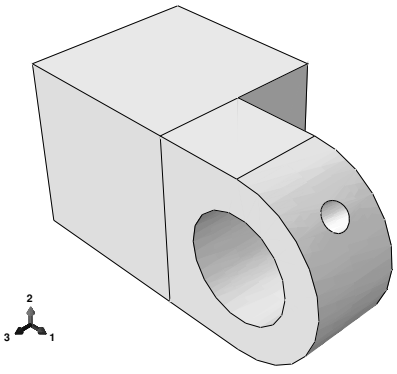


Figure 5: Isometric view of the first hinge.



Tip: After you rotate the part, use the cycle views tool  to step through the previous views (up to a maximum of eight) and to restore the original view.

13. Now that you have created the first part of your model, it is a good idea to save your model in a model database:
    a. From the main menu bar, select **File->Save**. The **Save Model Database As** dialog box appears.
    b. Type a name for the new model database in the **File Name** field, and click **OK**. You do not need to include the file extension; Abaqus/CAE appends `.cae` automatically to the file name.

       Abaqus/CAE stores the model database in a new file and returns to the Part module. The name of your model database appears in the main window title bar.

    If you find you need to interrupt this tutorial, you can save the model database at any time and exit Abaqus/CAE. You can then start a new Abaqus/CAE session and open the saved model database by selecting **Open Database** from the **Start Session** dialog box. The model database will contain any parts, materials, loads, etc. that you created, and you will be able to continue the tutorial.

Assigning section properties to the hinge part

To assign section properties to the hinge part, you will create a material, create a section that includes a reference to the material, and assign the section to the part or to a region of the part.

In this section:

- *Creating a material*
- *Defining a section*
- *Assigning the section*

Creating a material

You will create a material named `Steel` that has a Young's modulus of 209 GPa and a Poisson's ratio of 0.3.

1. In the Model Tree, double-click the **Materials** container to create a new material.
   The **Edit Material** dialog box appears.
2. Name the material `Steel`.
3. From the editor's menu bar, select **Mechanical**->**Elasticity**->**Elastic**.
   Abaqus/CAE displays the **Elastic** data form.
4. In the respective fields in the **Elastic** data form, type a value of `209.E9` for Young's modulus and a value of `0.3` for Poisson's ratio.
5. Click **OK** to exit the material editor.

Defining a section

Next, you will create a section that includes a reference to the material `Steel`.

1. In the Model Tree, double-click the **Sections** container to create a section.

   The **Create Section** dialog box appears.

2. In the **Create Section** dialog box:

   a. Name the section `SolidSection`.

   b. In the **Category** list, accept **Solid** as the default selection.

   c. In the **Type** list, accept **Homogeneous** as the default selection, and click **Continue**.

   The section editor appears.

3. In the editor, accept `Steel` as the material selection and click **OK**.

   If you had defined other materials, you could click the arrow next to the **Material** text box to see a list of available materials and to select the material of your choice.

Assigning the section

You will now assign the section `SolidSection` to the hinge part.

1. In the Model Tree, expand the **Hinge-hole** item underneath the **Parts** container and double-click **Section Assignments** in the list that appears.
2. Drag a rectangle around the hinge piece to select the entire part.

   Abaqus/CAE highlights all the regions of the part.
3. Click mouse button 2 to indicate that you have finished selecting the regions to be assigned the section.

   The **Edit Section Assignment** dialog box appears containing a list of existing sections. `SolidSection` is selected by default since there are no other sections currently defined.
4. In the **Edit Section Assignment** dialog box, accept the default selection of `SolidSection`, and click **OK**.

   Abaqus/CAE assigns the section to the part and colors the entire part aqua to indicate that the region has a section assignment.

Creating and modifying a second hinge piece

The model contains a second hinge piece similar to the first except that the lubrication hole is not present. You will create a copy of the first hinge piece and delete the features that form the lubrication hole.

In this section:

- *Copying the hinge*
- *Modifying the copy of the hinge*

Copying the hinge

First you will create an exact copy of the hinge piece.

1. In the Model Tree, click mouse button 3 on **Hinge-hole** underneath the **Parts** container and select **Copy** from the menu that appears.

   Abaqus/CAE displays the **Part Copy** dialog box.

2. In the text box in the **Part Copy** dialog box, type `Hinge-solid`, and click **OK**.

   Abaqus/CAE creates a copy of the hinge piece and names the copy **Hinge-solid**. The copy of the hinge piece includes the section from the original hinge piece.

Modifying the copy of the hinge

Now you will create a solid hinge piece by deleting the features that form the lubrication hole.

1. In the Model Tree, double-click **Hinge-solid** underneath the **Parts** container to make it current.

   Abaqus/CAE displays the part in the current viewport. Look at the viewport title bar to see which part is being displayed.

2. Expand the **Features** container underneath **Hinge-solid**.
3. Click mouse button 3 on **Datum pt-1** in the list of part features.

   Abaqus/CAE highlights the point, as shown in *Figure 1*.

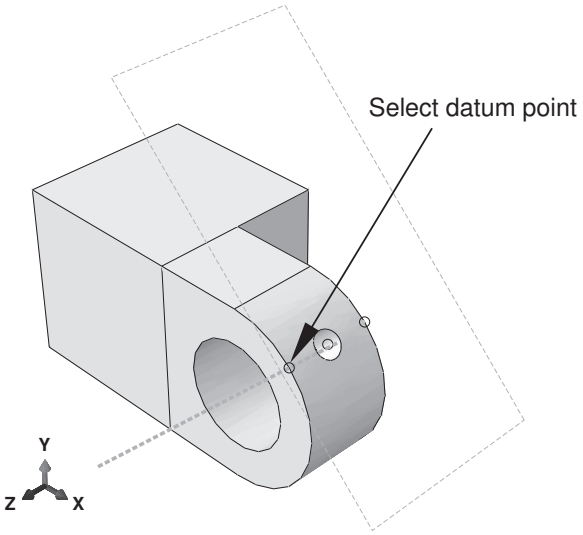


Figure 1: Delete the datum point and its children.

4. From the menu that appears, select **Delete**. When you delete a selected feature, Abaqus/CAE asks whether you also want to delete any features that depend on the feature being deleted. The feature being deleted is called the "parent" feature, and its dependent features are called "children." Abaqus/CAE highlights all the features that it will delete if the parent feature is deleted. From the buttons in the prompt area, click **Yes** to delete the datum point and all its children.

   Abaqus/CAE deletes the datum point. Because they were dependent on the datum point, Abaqus/CAE also deletes the datum axis, the datum plane, and the lubrication hole.

   Important: You cannot recover deleted features; however, you can temporarily remove a feature by suppressing it.

Creating the pin

The final assembly consists of instances of the two hinge pieces that are free to rotate about a pin. You will model the pin as a three-dimensional, revolved analytical rigid surface.
First you create the pin and assign the rigid body reference point; then you constrain the pin by applying constraints to this rigid body reference point.

In this section:

- *Creating the pin*
- *Assigning the rigid body reference point*

Creating the pin

You will now create the pin—a three-dimensional, revolved analytical rigid surface.

1. In the Model Tree, double-click the **Parts** container to create a new part.

   The **Create Part** dialog box appears.

2. Name the part `Pin`. Choose a three-dimensional body as before, but change the type to **Analytical rigid** and the base feature shape to **Revolved shell**.
3. Accept the approximate size of `0.2`, and click **Continue**.

   The Sketcher starts and displays the axis of revolution as a green dashed line with a fixed position constraint; your sketch cannot cross this axis.

4. From the Sketcher toolbox, select the connected lines tool . Sketch a vertical line to the right of the axis.
5. Dimension the horizontal distance from the line to the axis, and change the distance to `0.012`.
6. Dimension the vertical length of the line, and change the length to `0.06`.
7. Click mouse button 2 to exit the Sketcher.

   The sketch and the resulting shaded part are shown in *Figure 1*.

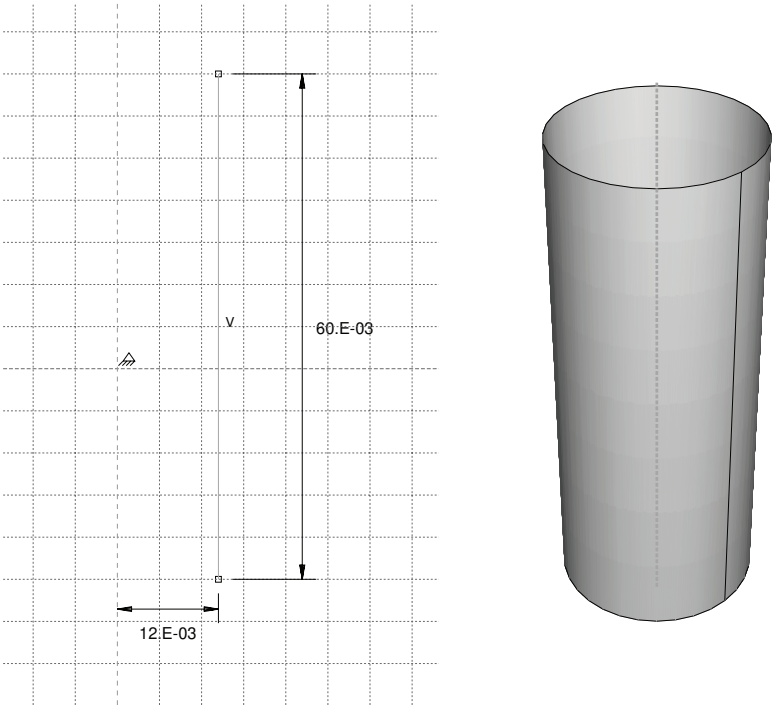


Figure 1: Create the pin by revolving an analytical rigid surface about an axis.

Assigning the rigid body reference point

You need to assign a rigid body reference point to the pin. Because you will not assign mass or rotary inertia to the pin, the rigid body reference point can be placed anywhere in the viewport. You use the Load module to apply constraints to the reference point or to define its motion. Motion or constraints that you apply to the rigid body reference point are applied to the entire rigid surface.

You can either select the reference point from the part in the viewport, or you can enter its coordinates. For the tutorial you will select the reference point from the viewport, as shown in *Figure 1*.

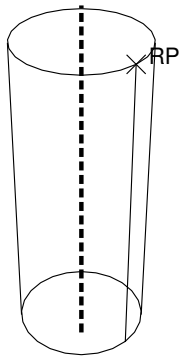


Figure 1: Create a rigid body reference point on the pin.

1. From the main menu bar, select **Tools->Reference Point**.
2. Select one of the vertices on the circumference of the pin.
   Abaqus/CAE labels the vertex **RP** to indicate that the reference point has been assigned to it.

Assembling the model

Your next task is to create instances of your parts.
A part instance can be thought of as a representation of the original part; an instance is not a copy of a part. You can then position these part instances in a global coordinate system to create the assembly.

An instance maintains its association with the original part. If the geometry of a part changes, Abaqus/CAE automatically updates all instances of the part to reflect these changes. You cannot edit the geometry of a part instance directly. The assembly can contain multiple instances of a single part; for example, a rivet that is used repeatedly in a sheet metal assembly.

An instance may be classified as independent or dependent. Independent part instances are meshed individually, while the mesh of a dependent part instance is associated with the mesh of the original part. Part meshing is discussed further in *Meshing the assembly*. By default, part instances are dependent.

When you create a part instance, Abaqus/CAE positions it so that the origin of the sketch that defined the base feature overlays the origin of the assembly's global coordinate system. In addition, the sketch plane is aligned with the *X*–*Y* plane of the global coordinate system.

When you create the first part instance, the Assembly module displays a graphic indicating the origin and the orientation of the global coordinate system. You can use this graphic to help you decide how to position a selected instance relative to the global coordinate system. For the tutorial you will keep the hinge with the lubrication hole fixed and move the second hinge and the pin relative to it.

For this tutorial you will create the following:

- An instance of the hinge piece with the lubrication hole—`Hinge-hole`.
- An instance of the hinge piece with the lubrication hole removed—`Hinge-solid`.
- An instance of the pin—`Pin`.

In this section:

- *Creating an instance of the first hinge*
- *Creating an instance of the second hinge*
- *Positioning the solid hinge piece*
- *Creating and positioning an instance of the pin*

Creating an instance of the first hinge

You will create an instance with a lubrication hole.

1. In the Model Tree, expand the **Assembly** container. Then double-click **Instances** in the list that appears to create a new part instance.

   The **Create Instance** dialog box appears containing a list of all the parts in the current model—the two hinge pieces and the pin in this example.

2. In the dialog box, select `Hinge-hole`.

   Abaqus/CAE displays a temporary image of the selected part.

3. In the dialog box, click **Apply**.

   Note: What is the difference between the **OK** and **Apply** buttons? When you click **OK**, the **Create Instance** dialog box closes once the part is instanced. When you click **Apply**, the **Create Instance** dialog box remains open while you create the instance and is available for you to create the next instance. Click **OK** if you want to create only a single part instance; click **Apply** if you want to create several part instances before moving on to a new procedure.

   Abaqus/CAE creates a dependent instance of the hinge piece and displays a graphic indicating the origin and orientation of the global coordinate system. Abaqus/CAE names the instance `Hinge-hole-1` to indicate that it is the first instance of a part called `Hinge-hole`.

   Note: The default position of a part instance is such that the origin and the *X*- and *Y*-axes of the sketch of the base feature align with the origin and the *X*- and *Y*-axes of the global coordinate system. For example, the base feature of the hinge piece is the original cube you created. Abaqus/CAE positions instances of the hinge piece so that the origin of the cube sketch is located at the origin of the global coordinate system and the *X*- and *Y*-axes align.

Creating an instance of the second hinge

You will now create an instance of the solid hinge piece.
To separate the solid hinge piece from the instance of the hinge piece with the lubrication hole, you ask Abaqus/CAE to offset the new instance along the *X*-axis.

1. From the **Create Instance** dialog box, toggle on **Auto-offset from other instances**.

   The auto-offset function prevents new part instances from overlapping existing instances.

2. From the **Create Instance** dialog box, select **Hinge-solid** and click **OK**.

   Abaqus/CAE closes the dialog box, creates the new dependent instance, and applies an offset along the *X*-axis that separates the two hinges, as shown in *Figure 1*. (For clarity the datum geometry has been removed from the shaded view in *Figure 1* and subsequent figures by selecting **View->Assembly Display Options->Datum**.)

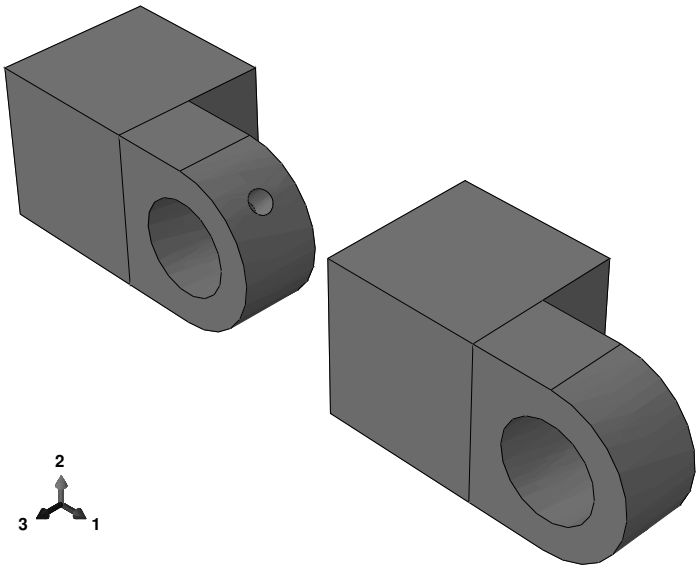


Figure 1: Create an instance of each hinge piece, and apply an offset to position them in the viewport.

Positioning the solid hinge piece

In addition to the simple translate and rotate procedures, the Assembly module provides a set of tools that allow you to position a selected part instance by defining the relationship between selected faces or edges. You can select a face (or an edge) of the instance to move, called the movable part instance, and a face (or an edge) of the instance that remains fixed, called the fixed part instance, and choose one of the following position constraints:

Parallel Face

The movable instance moves until the two selected faces are parallel.

Face to Face

The movable instance moves until the two selected faces are parallel and a specified clearance from each other.

Parallel Edge

The movable instance moves until the two selected edges are parallel.

Edge to Edge

The movable instance moves until the two selected edges are colinear or a specified distance from each other.

Coaxial

The movable instance moves until the two selected faces are coaxial.

Coincident Point

The movable instance moves until the two selected points are coincident.

Parallel CSYS

The movable instance moves until the two selected datum coordinate systems are parallel.

Abaqus/CAE stores position constraints as features of the assembly, and they can be edited, deleted, and suppressed. In contrast, translations and rotations are not stored and do not appear in the list of features. Although position constraints are stored as features, they have no knowledge of each other; as a consequence, a new position constraint may override a previous position constraint.

In this example you will move the solid hinge piece while the hinge piece with the lubrication hole will remain fixed. You will apply three types of position constraints to position the two hinge pieces correctly.

1. First, constrain the solid hinge piece so that the two flanges face each other. From the main menu bar, select **Constraint->Face to Face**.
2. From the movable part instance, select the face of the solid hinge piece shown in *Figure 1*.

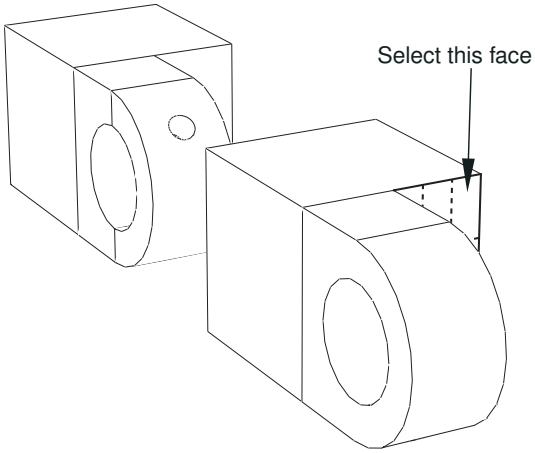


Figure 1: Select a face on the movable part instance.

3. From the fixed part instance, select the face of the hinge piece with the lubrication hole shown in *Figure 2*. Abaqus/CAE highlights the face on the movable part instance in red and the face on the fixed part instance in magenta.

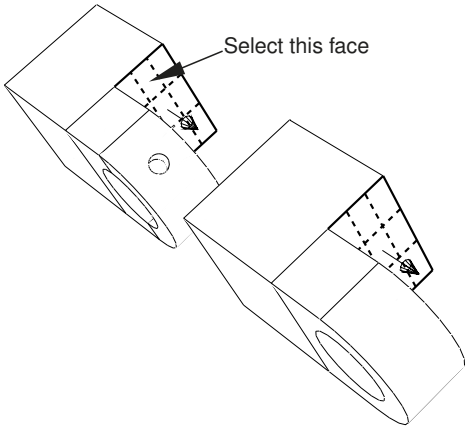


Figure 2: Select a face on the fixed instance.

Abaqus/CAE displays red arrows on each selected face; the movable instance will be positioned so that the arrows point in the same direction. You can change the direction of the arrow on the movable instance if necessary.

4. From the prompt area, click **Flip** to change the direction of the arrow. Click **OK** when the arrows point toward each other.
5. In the text box that appears in the prompt area, type the clearance (0.04) that will remain between the two parts, as measured along the normal to the selected face of the fixed part, and press **[Enter]**.

   Abaqus/CAE rotates the solid hinge piece so that the two selected faces are parallel to each other and 0.04 meters apart, as shown in *Figure 3*.

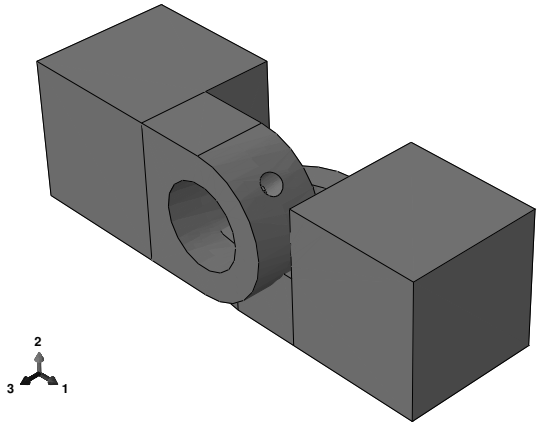


Figure 3: Position 1: Constrain the flange of the solid hinge piece to face the flange of the hinge piece with the lubrication hole.

The two pieces overlap because the position of the solid hinge piece is not fully determined by the position constraint you have applied. You will need to apply two more position constraints to obtain the desired position.

6. Next, align the two flange holes. From the main menu bar, select **Constraint->Coaxial**.
7. Select the flange hole on the solid hinge piece, as shown in *Figure 4*. (You may find it helpful to display the wireframe view of the two pieces.)

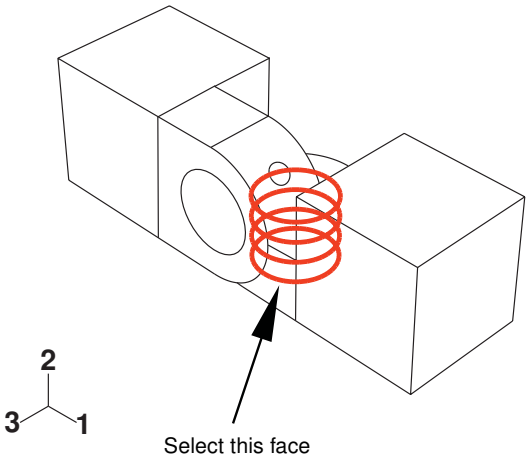


Figure 4: Select a cylindrical face on the movable instance.

8. Select the flange hole on the hinge piece with the lubrication hole, as shown in *Figure 5*.
   Abaqus/CAE displays red arrows on each selected face.

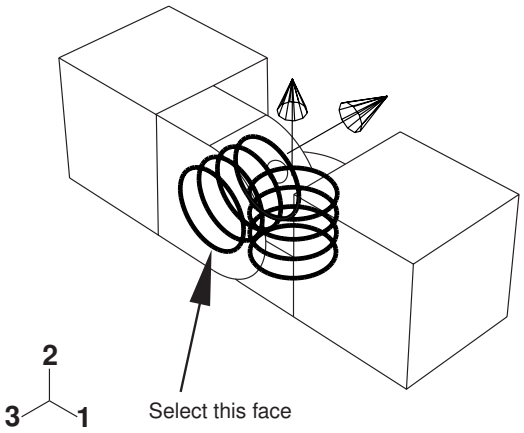


Figure 5: Select a cylindrical face on the fixed instance.

9. From the prompt area, click **Flip** to change the direction of the arrow on the movable part instance. Click **OK** when the arrow points downward.

   Abaqus/CAE positions the two hinge pieces so that the two flange holes are coaxial.

10. Use the rotate tool to look at the top view of the two pieces. Notice that the two flanges are now overlapping, as shown in *Figure 6*.

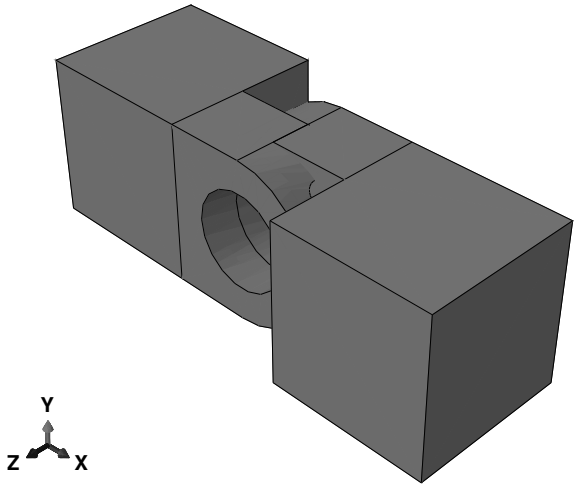


Figure 6: Position 2: Constrain the two flange holes to lie along the same axis.

11. Finally, add a constraint to eliminate the overlap between the two flanges. From the main menu bar, select **Constraint->Edge to Edge**.
12. Select the straight edge on the solid hinge piece shown in *Figure 7*.

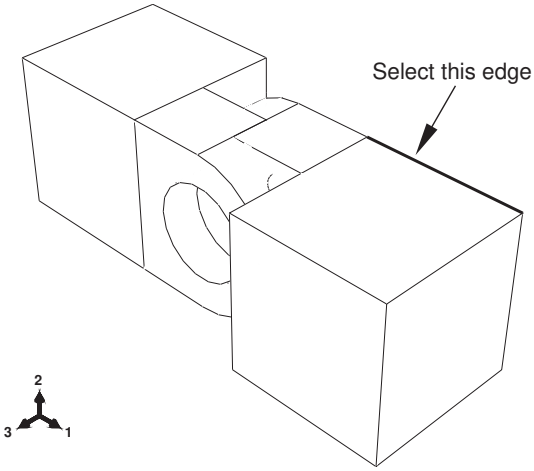


Figure 7: Select a straight edge on the movable instance.

13. Select the corresponding edge of the hinge piece with the lubrication hole, as shown in *Figure 8*.

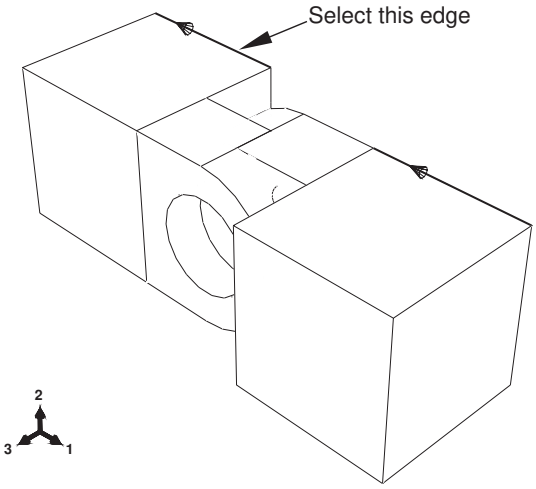


Figure 8: Select a straight edge on the fixed instance.

Abaqus/CAE displays red arrows on each selected face.

14. If necessary, flip the arrows so they point in the same direction; then click **OK** to apply the constraint.

Abaqus/CAE positions the two hinge pieces so that the two selected edges are colinear, as shown in *Figure 9*.

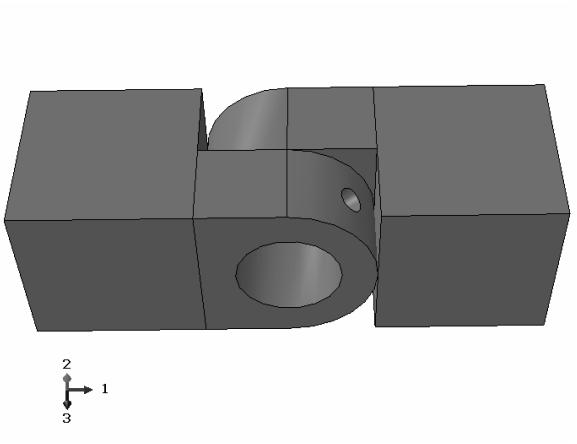


Figure 9: Final position: Constrain an edge of each hinge piece to lie along the same line.

Creating and positioning an instance of the pin

You will now create an instance of the pin and position it symmetrically in the flange holes using constraints and translation vectors. To define the translation vector, you can select vertices from the assembly or you can enter the coordinates. You can determine the translation vector using the **Query** tool.

1. In the Model Tree, double-click **Instances** underneath the **Assembly** container.
2. In the **Create Instance** dialog box, toggle off **Auto-offset from other instances** and create an instance of the pin.
3. Constrain the pin to lie along the same axis as the two flange holes. Use the **Constraint->Coaxial** menu as you did when you aligned the two flange holes in the previous section. (You can select either of the flange holes as the cylindrical surface of the fixed instance, and the direction of the arrows is not important.)

   Abaqus/CAE will position the pin as shown in *Figure 1*.

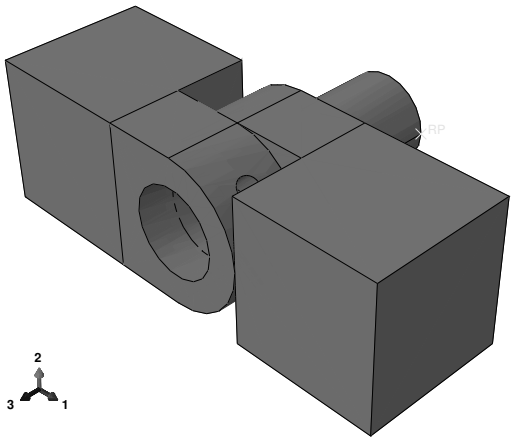


Figure 1: Align the pin to be coaxial with the two flange holes.

4. From the main menu bar, select **Tools->Query**.

   The **Query** dialog box appears.
5. Select **Distance** from the list of **General Queries**.
6. The **Distance** query allows you measure the *X*-, *Y*-, and *Z*-components of the vector connecting two selected points. You need to determine the distance between the end of the pin and the hinge containing the lubrication hole; the two points to select are illustrated in *Figure 2*.

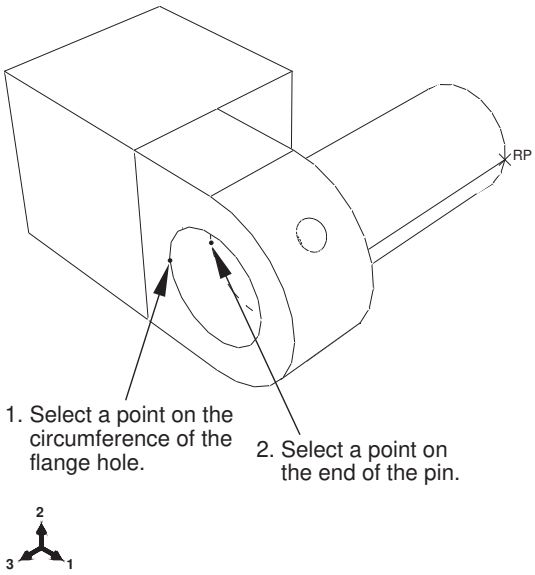


Figure 2: Determining the position of the pin.

a. To define one end of the vector, select a point on the circumference of the hole in the flange containing the lubrication hole.

b. To define the other end of the vector, select the vertex on the pin that is inside the hinge containing the lubrication hole.

Abaqus/CAE displays the vector distance between the two selected points along with the *X*-, *Y*-, and *Z*-components of the vector in the message area. You will translate the pin along the *Z*-axis; the *Z*-component of the distance is 0.01 meters. You want to position the pin symmetrically between the hinges, so you will translate it 0.02 meters.

7. From the main menu bar, select **Instance->Translate**.

8. Select the pin as the part instance to move, and click **Done** to indicate that you have finished selecting instances to move.

9. In the text boxes in the prompt area, enter a start point for the translation vector of `0,0,0` and an end point of `0,0,0.02`.

Abaqus/CAE translates the pin a distance of 0.02 along the *Z*-axis and displays a temporary image of the new position of the pin.

Note:

If the position of a temporary image (colored red) is not correct, you can use the buttons in the prompt area to correct the problem. Click either the **Cancel** button ( ) to cancel the procedure or the **Previous** button ( ) to step back through the procedure.

10. From the prompt area, click **OK**.

The finished assembly is shown in *Figure 3*.

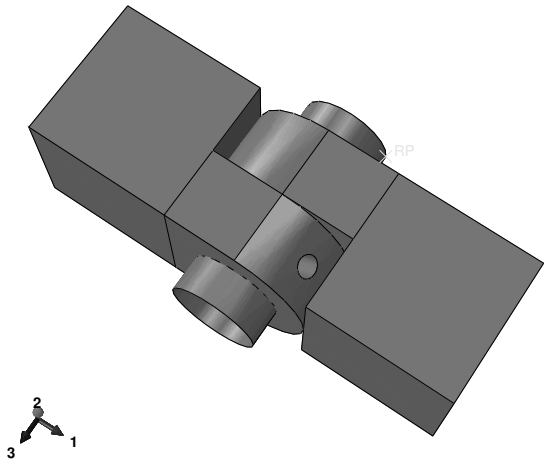


Figure 3: Shaded view of the finished assembly.

11. Before proceeding, convert all position constraints to absolute positions. From the main menu bar, select **Instance->Convert Constraints**. Select all part instances, and click **Done** in the prompt area.

Defining analysis steps

Before you apply loads or boundary conditions to the model or define contact within the model, you must define the different steps in the analysis. Once the steps are created, you can specify in which steps loads, boundary conditions, and interactions should be applied.

When you create a step, Abaqus/CAE selects a default set of output variables corresponding to the analysis procedure and selects a default rate at which the variables are written to the output database. In this tutorial you will edit the default output frequency for the first step and edit the list of default output variables for the second step.

In this section:

- *Creating the analysis steps*
- *Requesting output*
- *Selecting a degree of freedom to monitor*

Creating the analysis steps

The analysis that you perform on the hinge model will consist of an initial step and two general analysis steps:

- In the initial step you apply boundary conditions to regions of the model and define contact between regions of the model.
- In the first general analysis step you allow contact to become established.
- In the second general analysis step you modify two of the boundary conditions applied to the model and apply a pressure load to one of the hinge pieces.

Abaqus/CAE creates the initial step by default, but you must create the two analysis steps.

1. In the Model Tree, double-click the **Steps** container to create a new step.

   The **Create Step** dialog box appears.

2. In the **Create Step** dialog box:

   a. Name the step `Contact`.

   b. Accept the default procedure type (**Static, General**), and click **Continue**.

   The step editor appears.

3. In the **Description** field, type `Establish contact`.
4. Click the **Incrementation** tab, and delete the value of `1` that appears in the **Initial** text field. Type a value of `0.1` for the initial increment size.
5. Click **OK** to create the step and to exit the editor.

   The `Contact` step appears underneath the **Steps** container in the Model Tree.

6. Use the same technique to create a second general, static step named `Load`. Enter `Apply load` in the description field and an initial increment size of `0.1`.

   The `Load` step appears underneath the **Steps** container in the Model Tree.

Requesting output

You use field output requests to request output of variables that should be written at relatively low frequencies to the output database from the entire model or from a large portion of the model. Field output is used to generate deformed shape plots, contour plots, and animations from your analysis results. Abaqus/CAE writes every component of the variables to the output database at the selected frequency.

You use history output requests to request output of variables that should be written to the output database at a high frequency from a small portion of the model; for example, the displacement of a single node. History output is used to generate *X–Y* plots and data reports from your analysis results. When you create a history output request, you must select the individual components of the variables that will be written to the output database.

The default field output variables for the `Contact` and `Load` steps include the following:

- **S** (Stress components)
- **PE** (Plastic strain components)
- **PEEQ** (Equivalent plastic strain)
- **PEMAG** (Plastic strain magnitude)
- **LE** (Logarithmic strain components)
- **U** (Translations and rotations)
- **RF** (Reaction forces and moments)
- **CF** (Concentrated forces and moments)
- **CSTRESS** (Contact stresses)
- **CDISP** (Contact displacements)

By default, Abaqus/CAE writes the default field output variables from a static, general procedure to the output database after every increment of a step. In the following procedure you will change the output frequency during the `Contact` step so that data are written to the output database once—at the last increment of the step. In addition, you will delete the request for CDISP during the `Load` step, since it is not needed for postprocessing.

1. In the Model Tree, click mouse button 3 on the **Field Output Requests** container and select **Manager** from the menu that appears.

   The **Field Output Requests Manager** appears. The **Field Output Requests Manager** is a step-dependent manager. The types of objects that appear in step-dependent managers are those that you can create, modify, and deactivate in particular analysis steps. Step-dependent managers display information concerning the history of each object listed in the manager. In this example Abaqus/CAE named the default field output request that you created in the `Contact` step **F-Output-1**. In addition Abaqus/CAE propagated the output request into the `Load` step. For more information, see *Managing objects*.

2. From the **Field Output Requests Manager**, select the **F-Output-1** output request in the `Contact` step. From the buttons on the right side of the manager, click **Edit**.

   The **Edit Field Output Request** editor appears for the `Contact` step.

3. Select **Last increment** as the output frequency to generate output only during the last increment of the step.
4. Select **Exterior only** to restrict field output to nodes and elements that belong to the exterior of the model.
5. Click **OK** to modify the output request.
6. From the **Field Output Requests Manager**, select the **F-Output-1** output request in the `Load` step and click **Edit**.

The **Edit Field Output Request** editor appears for the `Load` step.

7. Set the output frequency to `1` to generate output during every increment of the step.
8. From the list of output categories, click the arrow to the left of **Contact**.

   A list of the contact output variables available appears along with a description of each.
9. Click the check box next to **CDISP** to deselect this variable for output.

   The check box next to **Contact** remains light gray with a dark gray check mark to indicate that not all variables in this category will be output. The **Edit Field Output Request** editor also indicates the following:

   - Output will be generated for the whole model.
   - Output will be generated at default section points.
   - Output will include local coordinate transformations (when available).
10. Click **OK** to modify the output request.

    In the **Field Output Requests Manager** the status of the output request changes to **Modified** for the `Load` step.
11. At the bottom of the **Field Output Requests Manager**, click **Dismiss** to close the dialog box.

Selecting a degree of freedom to monitor

You can define sets that contain only selected portions of your model. Once you create a set, you can use it to perform the following tasks:

- Assign section properties in the Property module.
- Create contact pairs with contact node sets and surfaces in the Interaction module.
- Define loads and boundary conditions in the Load module.
- Request output to either the output database or the status file from specific regions of the model in the Step module. Output to the status file is also reported back to the Job module in the form of a continuously updated *X–Y* plot.
- Display results for specific regions of the model in the Visualization module.

In this example you will define a set consisting of a single point. You will then be able to monitor the results for one degree of freedom at that point when you submit your job for analysis later in this tutorial.

1. In the Model Tree, expand the **Assembly** container and double-click the **Sets** item.
   The **Create Set** dialog box appears.
2. Name the set `Monitor`, and click **Continue**.
3. Select the vertex of the solid hinge piece shown in *Figure 1*.

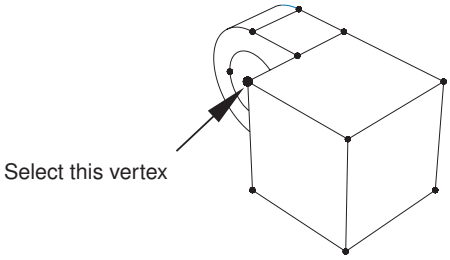


Figure 1: Monitor a degree of freedom on the solid hinge piece.

4. Click **Done** to indicate that you have finished selecting the geometry for the set.
   Abaqus/CAE creates a node set with the name `Monitor` that contains the node you selected.
5. From the main menu bar of the Step module, select **Output->DOF Monitor**.
   The **DOF Monitor** dialog box appears.
6. Toggle on **Monitor a degree of freedom throughout the analysis**.
7. Click , then click **Points** in the prompt area and choose the set **Monitor** from the **Region Selection** dialog box.
8. Type `1` in the **Degree of freedom** text field, and click **OK**.

Creating surfaces to use in contact interactions

Now you will define contact between regions of the model.

In this problem general contact will be used. Typically, surfaces need not be explicitly defined for general contact with the exception of surfaces associated with analytical rigid surfaces. You will define the following surface:

- A surface named `Pin` that includes the outside surface of the pin.

You will find it helpful to display only one part instance at a time while you select the surfaces to be defined.

Hide part instances in the assembly

1. In the Model Tree, expand the **Instances** container underneath the **Assembly**.
2. Select the hinge pieces, and click mouse button 3.
3. Choose **Hide** from the menu that appears.

   The hinge pieces disappear from the view.

Define a surface on the pin

1. In the Model Tree, expand the **Assembly** container and double-click the **Surfaces** item.

   The **Create Surface** dialog box appears.
2. In the dialog box, name the surface `Pin` and click **Continue**.
3. In the viewport, select the pin.
4. Click mouse button 2 in the viewport to indicate that you have finished selecting regions for the surface.

   Each side of the hollow cylinder representing the pin has a different color associated with it. In *Figure 1* the outside of the pin is colored brown and the inside of the pin is colored purple. The colors may be reversed on your model, depending on how you created the original sketch for the pin.

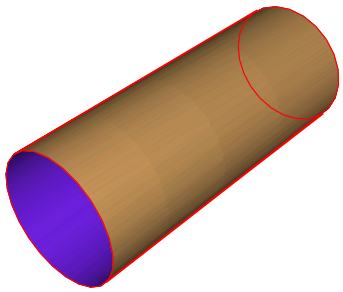


Figure 1: Select the region to be defined as the surface `Pin`.

5. You must choose whether the surface consists of the inside or the outside of the cylinder. The outside surface contacts the two hinges and is the desired choice. From the buttons in the prompt area, click the color (**Brown** or **Purple**) associated with the outside surface.

   Abaqus/CAE creates the desired surface called `Pin` and displays it underneath the **Surfaces** container in the Model Tree.

Defining contact between regions of the model

Interactions are objects that you create to model mechanical relationships between surfaces that are in contact or closely spaced.
Mere physical proximity of two surfaces on an assembly is not enough to indicate any type of interaction between the surfaces. A contact interaction is required. The general contact approach is used in this example.

A contact interaction requires a reference to an interaction property. Interaction properties are collections of information that help you to define certain types of interactions. You will create a mechanical interaction property that describes the tangential and normal behavior between all surfaces as frictionless. You will name this property `NoFric`.

In this section:

- *Creating an interaction property*
- *Defining a general contact interaction*

Creating an interaction property

In this procedure you will create a mechanical contact interaction property.

1. In the Model Tree, double-click the **Interaction Properties** container to create a contact property.

   The **Create Interaction Property** dialog box appears.

2. In the **Create Interaction Property** dialog box:
   a. Name the property `NoFric`.
   b. In the **Type** list, accept **Contact** as the default selection.
   c. Click **Continue**.

   The **Edit Contact Property** dialog box appears.

3. From the dialog box's menu bar, select **Mechanical->Tangential Behavior** and accept **Frictionless** for the friction formulation.
4. Click **OK** to save your settings and to close the **Edit Contact Property** dialog box.

Defining a general contact interaction

In this section you will create a general contact interaction that will refer to the interaction property that you just created.

1. In the Model Tree, double-click the **Interactions** container.
2. In the **Create Interaction** dialog box, set the step to `Initial` and choose **General contact (Standard)** as the type.
3. Click **Continue**.
4. In the interaction editor, select `No Fric` from the list of available **Global property assignment** options.
5. Click **OK** to complete the contact definition.

Applying boundary conditions and loads to the assembly

The example demonstrates how to apply boundary conditions and loads to the assembly.

In this section:

- *Application of boundary conditions and load*
- *Constraining the hinge piece with the lubrication hole*
- *Constraining the pin*
- *Modifying the boundary condition applied to the pin*
- *Constraining the solid hinge piece*
- *Applying a load to the solid hinge*

Application of boundary conditions and load

In this example you will apply boundary conditions and loads to the hinge model.

You will apply the following boundary conditions and load:

- A boundary condition called `Fixed` that constrains all degrees of freedom at the end of the hinge piece with the lubrication hole, as shown in *Figure 1*.

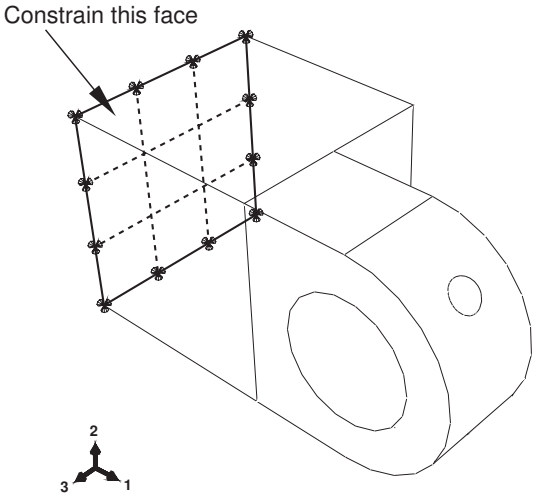


Figure 1: One end of the hinge is constrained.

- A boundary condition called `NoSlip` that constrains all degrees of freedom of the pin while contact is established during the first analysis step. You will modify this boundary condition in the second analysis step (the step in which the load is applied) so that degrees of freedom 1 and 5 are unconstrained. *Figure 2* illustrates this boundary condition applied at the reference point.

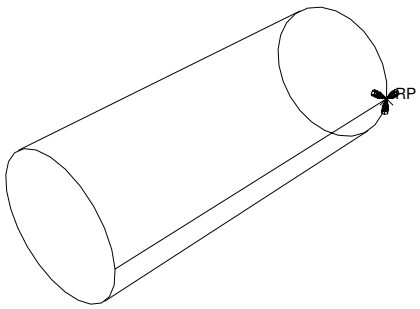


Figure 2: The pin is constrained.

- A boundary condition called `Constrain` that constrains all degrees of freedom of a point on the solid hinge piece during the first analysis step. You will modify this boundary condition in the second analysis step so that degree of freedom 1 is unconstrained when the load is applied.
- A load called `Pressure` that you apply to the end of the solid hinge piece during the second analysis step. *Figure 3* illustrates the constraint and the pressure load applied to the solid hinge.

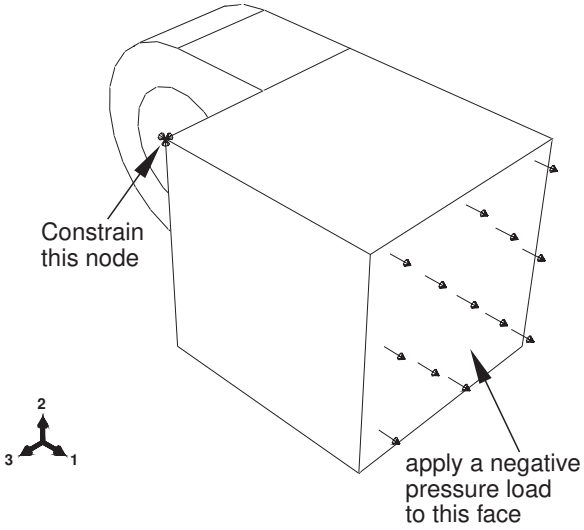


Figure 3: The second hinge is constrained and loaded.

Constraining the hinge piece with the lubrication hole

You will apply a boundary condition to the face at the end of the hinge piece with the lubrication hole to fix the hinge piece in place during the analysis.

1. In the Model Tree, click mouse button 3 on the **BCs** container and select **Manager** from the menu that appears.

   The **Boundary Condition Manager** dialog box appears.

2. In the **Boundary Condition Manager**, click **Create**.

   The **Create Boundary Condition** dialog box appears.

3. In the **Create Boundary Condition** dialog box:

   a. Name the boundary condition `Fixed`.

   b. Select `Initial` from the list of steps.

   c. Accept **Mechanical** as the default **Category** selection.

   d. Select **Displacement/Rotation** as the type of boundary condition for the selected step.

   e. Click **Continue**.

      The **Region Selection** dialog box appears.

   f. From the right side of the prompt area, click **Select in Viewport** to select the object directly from the viewport.

      The **Region Selection** dialog box closes.

4. Select the gridded face shown in *Figure 1* as the region where the boundary condition will be applied. You may need to rotate the view in order to select this face.

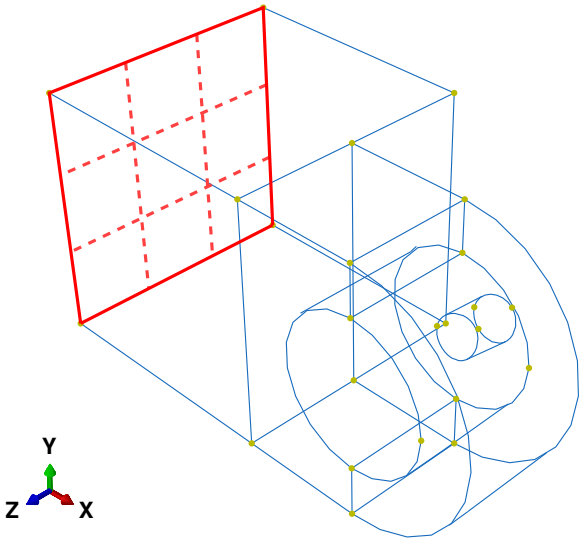


Figure 1: Apply a boundary condition to the end of the hinge piece with the lubrication hole.

5. Click mouse button 2 to indicate that you have finished selecting regions.

   The **Edit Boundary Condition** dialog box appears.

6. In the dialog box:

a. Toggle on the buttons labeled **U1**, **U2**, and **U3** to constrain the end of the hinge in the 1-, 2-, and 3-directions. You do not need to constrain the rotational degrees of freedom of the hinge because solid elements (which have only translational degrees of freedom) will be used to mesh the hinge.

b. Click **OK** to close the dialog box.

The boundary condition that you just created appears in the **Boundary Condition Manager**, and arrows appear on the nodes of the face indicating the constrained degrees of freedom. The **Boundary Condition Manager** shows that the boundary condition remains active in all steps of the analysis.



Tip: To suppress the display of boundary condition arrows, click the **Attribute** tab in the **Assembly Display Options** dialog box to access the boundary condition display options.

Constraining the pin

In the first general step of the analysis you will establish contact between the two hinge pieces and between the hinge pieces and the pin. To fix the pin during this step, you must apply a boundary condition to the pin that constrains all its degrees of freedom.

1. In the **Boundary Condition Manager**, click **Create**.

   The **Create Boundary Condition** dialog box appears.

2. In the **Create Boundary Condition** dialog box:
   a. Name the boundary condition `NoSlip`.
   b. Accept `Initial` in the **Step** text field.
   c. Accept **Mechanical** as the default **Category** selection.
   d. Select **Displacement/Rotation** as the type of boundary condition for the selected step.
   e. Click **Continue**.

3. In the viewport, select the rigid body reference point on the pin as the region where the boundary condition will be applied.

4. Click mouse button 2 to indicate that you have finished selecting regions.

   The **Edit Boundary Condition** dialog box appears.

5. In the dialog box:
   a. Toggle on all the buttons to constrain all the degrees of freedom of the pin.
   b. Click **OK**.

      The new boundary condition appears in the **Boundary Condition Manager**.

Modifying the boundary condition applied to the pin

Objects that you can create and modify in certain steps—such as boundary conditions, loads, and interactions—have special managers that allow you to modify objects and change their status in different analysis steps.

In this section you will use the boundary condition manager to modify the boundary condition `NoSlip` so that translation in the 1-direction and rotation about the 2-axis are unconstrained during the loading step.

Currently the **Boundary Condition Manager** displays the names of the two boundary conditions that you have created as well as their status in each step: both boundary conditions are `Created` in the initial step and `Propagated` through the following analysis steps.

1. In the **Boundary Condition Manager**, click the cell labeled `Propagated` that lies in the row labeled `NoSlip` and in the column labeled `Load`, as shown in *Figure 1*. That cell becomes highlighted.

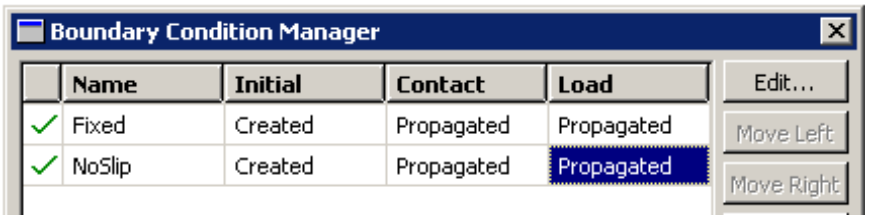


Figure 1: Select boundary conditions to edit in the Boundary Condition Manager.

2. On the right side of the manager, click **Edit** to indicate that you want to edit the `NoSlip` boundary condition in the `Load` step.

   The **Edit Boundary Condition** dialog box appears, and Abaqus/CAE displays a set of arrows on the model indicating where the boundary condition is applied and which degrees of freedom are constrained.

3. In the editor, toggle off the buttons labeled **U1** and **UR2** so that the pin is allowed to translate in the 1-direction and rotate about the 2-axis. Click **OK** to close the dialog box.

   In the **Boundary Condition Manager**, the status of the `NoSlip` boundary condition in the `Load` step changes to `Modified`.

Constraining the solid hinge piece

In the first analysis step, in which contact is established, you will constrain a single node of the solid hinge piece in all directions. These constraints, along with contact with the pin, are enough to prevent rigid body motion of the solid piece. In the second analysis step, in which the load is applied to the model, you will remove the constraint in the 1-direction.

1. Create a displacement boundary condition in the `Initial` step, and call it `Constrain`.
2. Apply the boundary condition to the vertex selected from the solid hinge piece, as shown in *Figure 1*.

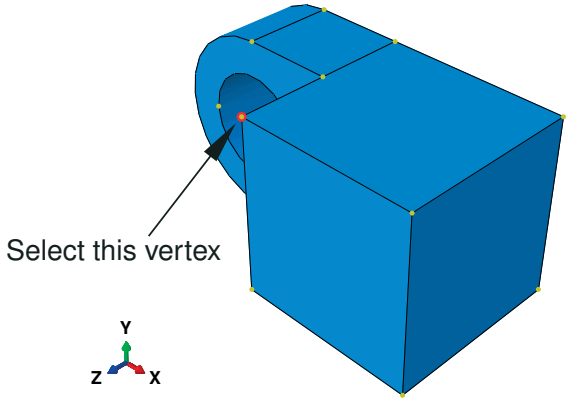


Figure 1: Apply a boundary condition to a vertex of the solid hinge piece.

3. Constrain the vertex in the 1-, 2-, and 3-directions.
4. In the `Load` step, release the constraint in the 1-direction.
5. When you have finished creating boundary conditions, click **Dismiss** to close the **Boundary Condition Manager**.

Applying a load to the solid hinge

Next, you apply a pressure load to the face at the end of the solid hinge. You apply the load in the 1-direction during the second analysis step.

1. In the Model Tree, double-click the **Loads** container to create a new load.

   The **Create Load** dialog box appears.

2. In the **Create Load** dialog box:
   a. Name the load `Pressure`.
   b. Accept `Load` as the default selection in the **Step** text field.
   c. From the **Category** list, accept **Mechanical** as the default selection.
   d. From the **Types for Selected Step** list, select **Pressure**.
   e. Click **Continue**.

3. In the viewport, select the face at the end of the solid hinge piece as the surface to which the load will be applied, as shown by the gridded surface in *Figure 1*.

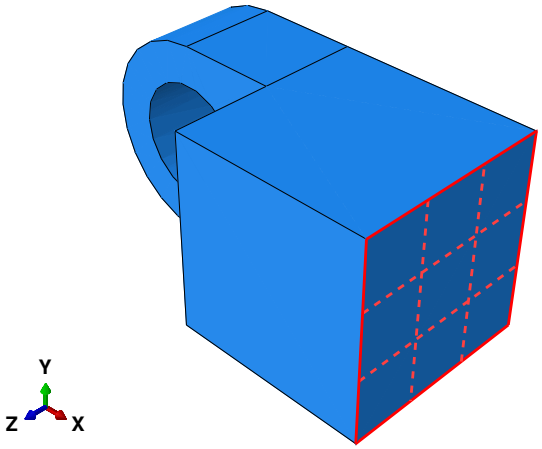


Figure 1: Apply a load to the solid hinge piece.

4. Click mouse button 2 to indicate that you have finished selecting regions.

   The **Edit Load** dialog box appears.

5. In the dialog box, enter a magnitude of `-1.E6` for the load, and click **OK**.

   Arrows appear on the face indicating the applied load. The arrows are pointing out of the face because you applied a negative pressure.

Meshing the assembly

Meshing the assembly is divided into the following operations:

- Making sure the part instances can be meshed and creating additional partitions where necessary.
- Assigning mesh attributes to the part instances.
- Seeding the part instances.
- Meshing the part instances.

In this section:

- *Deciding what needs to be partitioned*
- *Partitioning the flange with the lubrication hole*
- *Assigning mesh controls*
- *Assigning the Abaqus element type*
- *Seeding the part instances*
- *Meshing the assembly*

Deciding what needs to be partitioned

When you enter the Mesh module, Abaqus/CAE color codes regions of the model according to the methods it will use to generate a mesh:

- Green indicates that a region can be meshed using structured methods.
- Yellow indicates that a region can be meshed using sweep methods.
- Orange indicates that a region cannot be meshed using the default element shape assignment (hexahedral) and must be partitioned further. (Alternatively, you can mesh any model by assigning tetrahedral elements to the model and using the free meshing technique.)

For the tutorial Abaqus/CAE indicates that the hinge with the lubrication hole needs to be partitioned to be meshed using hexahedral-shaped elements. Specifically, areas surrounding the hole in the flange must be partitioned. The partitioned hinges are shown in *Figure 1*.

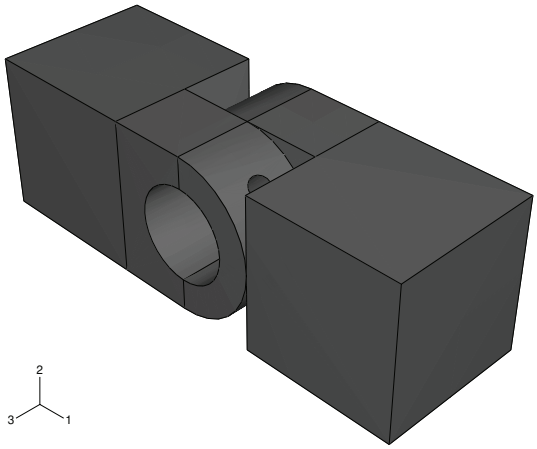


Figure 1: The partitioned hinges.

Use the following techniques to help you select faces and vertices during the partitioning process:

- Use a combination of the view manipulation tools, the display option tools in the View Options toolbar, and the tools in the Views toolbar to resize and reposition the model as necessary. (To display the Views toolbar, select **View->Toolbars->Views** from the main menu bar.)
- Toggle off the closest object tool  in the **Selection** toolbar to cycle through the possible selections using the **Next** and **Previous** buttons in the prompt area.
- You will probably find the 3D compass and/or the magnification tool  and the rotation tool  especially useful.
- When necessary, click the **Iso** tool in the **Views** toolbox to return the model to its original size and position in the viewport.
- Recall that part instances are classified as dependent by default. All dependent instances of a part must possess identical geometry (including partitions) and meshes. To satisfy this requirement, all partitions must be created in the original part and all mesh attributes must be assigned to the original part. You will need to examine the parts individually to determine what action (if any) needs to be taken to create a mesh using hexahedral elements.

Note:

The advantage of dependent part instances is that if you create multiple instances of the same part, you need only manipulate and mesh the original part; these features are automatically inherited by the dependent instances. Since you created only one instance of each part in this tutorial, you could have created independent part instances and worked with them just as easily. This would have allowed you to create partitions and assign mesh attributes at the assembly level rather than at the part level. You can make a dependent part instance independent by clicking mouse button 3 on its name underneath the **Instances** container in the Model Tree and selecting **Make independent**. In what follows, we assume the part instances remain dependent.

1. In the Model Tree, expand **Hinge-hole** underneath the **Parts** container and double-click **Mesh** in the list that appears.

   Note:

   If the part instance were independent, you would instead expand the instance name underneath the **Instances** container and click **Mesh** in the list that would appear.

   Abaqus/CAE displays the hinge piece with the lubrication hole. The cube portion of the hinge piece is colored green to indicate that it can be meshed using the structured meshing technique; the flange with the lubrication hole is colored orange to indicate that it needs to be partitioned to be meshed using hexahedral elements, as shown in *Figure 2*. The partitioning procedure is described in *Partitioning the flange with the lubrication hole*.

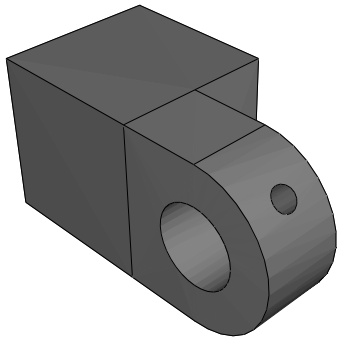


Figure 2: The flange with the lubrication hole cannot be meshed.

2. Use the **Object** field that appears in the context bar to display the solid hinge in the viewport. Abaqus/CAE displays the solid hinge. As before, the cube portion of the solid hinge piece is colored green to indicate that it can be meshed using the structured meshing technique. The flange without the lubrication hole is colored yellow to indicate that it can be meshed using a swept mesh.
3. Select the pin from the **Object** field in the context bar. Abaqus/CAE displays the pin in orange because it is an analytical rigid surface and cannot be meshed.

   Thus, the hinge piece with the lubrication hole needs to be partitioned to be meshed with hexahedral elements; the solid hinge and the pin require no further action.

Partitioning the flange with the lubrication hole

For Abaqus/CAE to mesh the flange with the lubrication hole, it must be partitioned.

The flange must be partitioned into the regions shown in *Figure 1*.

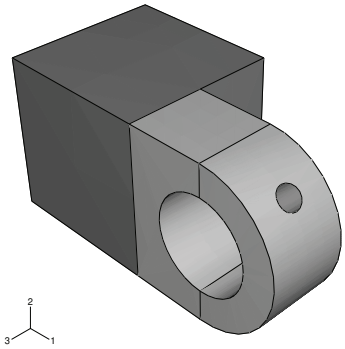


Figure 1: Shaded view of the partitioned flange.

1. Make the hinge piece with the lubrication hole current in the viewport.
2. From the main menu, select **Tools->Partition**.

   The **Create Partition** dialog box appears.
3. You want to partition the entire cell that makes up the flange. From the **Create Partition** dialog box, select **Cell** as the **Type** of partition and click **Define cutting plane** as the partition **Method**.
4. Select the flange of the hinge with the lubrication hole. Click **Done** to indicate you have finished selecting cells.

   Abaqus/CAE provides three methods for specifying the cutting plane:

   - Select a point and a normal. The cutting plane passes through the selected point, normal to the selected edge.
   - Select three non-colinear points. The cutting plane passes through each point.
   - Select an edge and a point along the edge. The cutting plane passes through the selected point, normal to the selected edge.

   The cutting plane need not be defined in the cell being partitioned. The plane extends infinitely and partitions the selected cell anywhere there is an intersection.
5. From the buttons in the prompt area, select **3 points**.

   Abaqus/CAE highlights points that you can select.
6. Select three points that cut the flanges in half with a vertical partition, as shown in *Figure 2*.



Tip: You may find it easier to select the desired points if you magnify, rotate, and pan the model to obtain a more convenient view.

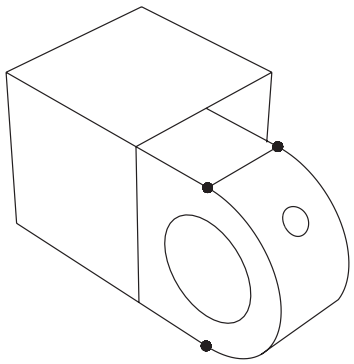


Figure 2: Select three points to use in partitioning the flanges.

7. From the prompt area, click **Create Partition**.

   Abaqus/CAE creates the desired partitions.

   The flange regions are colored yellow, indicating that no additional partitions are required to create a hexahedral mesh. Thus, the partitioning operation is complete.

8. Select **Assembly** in the **Object** field of the context bar to display the model assembly in the viewport. The model assembly with all the partitions is shown in *Figure 3*.

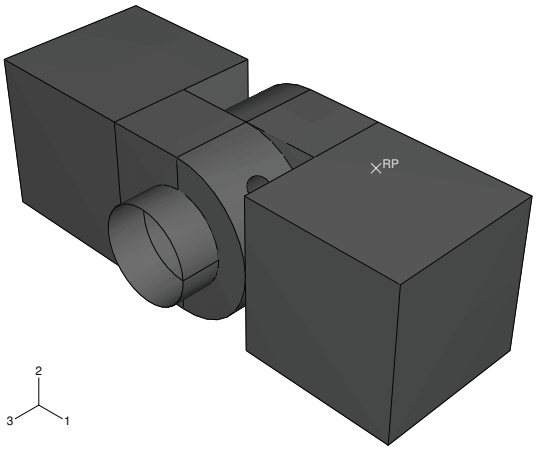


Figure 3: The partitioned model.

Tip: If you did not previously convert the positioning constraints to absolute positions, this partition may invalidate the coaxial constraints created earlier. If necessary, return to the Assembly module and define new coaxial constraints between the hinge pieces and between the hinge and the pin using the technique described earlier.

Assigning mesh controls

In this section you will use the **Mesh Controls** dialog box to examine the techniques that Abaqus/CAE will use to mesh the parts and the shape of the elements that Abaqus/CAE will generate.

You cannot mesh an analytical rigid surface. As a result you cannot apply mesh controls to an analytical rigid surface; neither can you seed it or assign an element type to it. Thus, you need only concern yourself with the hinge pieces. Since the instances are dependent on the original part definition, you must assign mesh attributes (controls, type, and seed size) to each hinge piece individually. For convenience, you will begin with the hinge piece with the lubrication hole.

1. Make the hinge piece with the hole current in the viewport. From the main menu bar, select **Mesh->Controls**.
2. Drag a square around the part to select all regions of the part, and click **Done** to indicate your selection is complete.

   The hinge piece appears red in the viewport to indicate that you have selected it, and Abaqus/CAE displays the **Mesh Controls** dialog box.
3. In the dialog box, accept **Hex** as the default **Element Shape** selection.
4. Select **Sweep** as the meshing technique that Abaqus/CAE will apply.
5. Select **Medial axis** as the meshing algorithm.
6. Click **OK** to assign the mesh controls and to close the dialog box.

   The entire hinge will appear in yellow, indicating that it will be meshed using the swept meshing technique.
7. Repeat the above steps for the solid hinge piece.

Assigning the Abaqus element type

In this section you will use the **Element Type** dialog box to examine the element types that are assigned to each part. For convenience, you will begin with the hinge piece with the lubrication hole.

1. Make the hinge piece with the hole current in the viewport. From the main menu bar, select **Mesh->Element Type**.
2. Select the hinge piece using the same technique described in the mesh controls procedure, and click **Done** to indicate your selection is complete.

   Abaqus/CAE displays the **Element Type** dialog box.
3. In the dialog box, accept **Standard** as the **Element Library** selection.
4. Accept **Linear** as the **Geometric Order** selection.
5. Accept **3D Stress** as the default **Family** of elements.
6. Click the **Hex** tab, and select **Reduced Integration** as the formulation if it is not already selected.

   A description of the default element type, C3D8R, appears at the bottom of the dialog box. Abaqus/CAE will now associate C3D8R elements with the elements in the mesh.
7. Click **OK** to assign the element type and to close the dialog box.
8. Click **Done** in the prompt area.
9. Repeat the above steps for the solid hinge piece.

Seeding the part instances

The next step of the meshing process is to seed each of the part instances. Seeds represent the approximate locations of nodes and indicate the target density of the mesh you would like to generate. You can select seeding based on the number of elements to generate along an edge or on the average element size, or you can bias seed distribution toward one end of an edge. For the tutorial you will seed the parts so that the hinge pieces have an average element size of 0.004. For convenience, you will begin with the hinge piece with the lubrication hole.

1. Make the hinge piece with the hole current in the viewport. From the main menu bar, select **Seed->Part**.
2. In the **Global Seeds** dialog box that appears, enter an approximate global element size of `0.004`, and click **OK**.

   Seeds appear on all the edges.

   Note:

   If you are using the Abaqus Learning Edition, using a global seed size of 0.004 results in a mesh that exceeds the model size limits of the product. Use a global seed size of 0.008 instead.

3. Repeat the above steps for the solid hinge piece.

You are now ready to mesh the parts.

Meshing the assembly

In this section you will mesh the parts. For convenience, you will begin with the hinge piece with the lubrication hole.

1. Make the hinge piece with the hole current in the viewport. From the main menu bar, select **Mesh->Part**.
2. Click **Yes** in the prompt area to create the mesh.
   Abaqus/CAE meshes the part.
3. Repeat the above steps for the solid hinge piece.

The meshing operations are now complete. Display the model assembly in the viewport to see the final mesh, as illustrated in *Figure 1*.

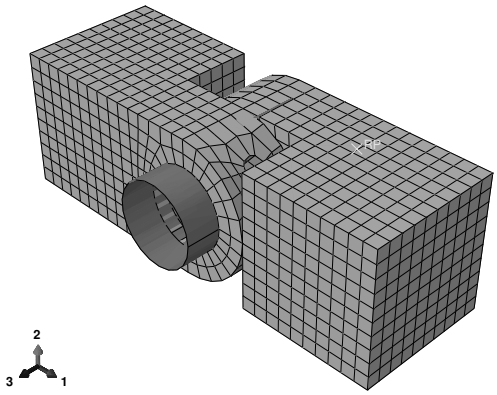


Figure 1: Final view of the meshed model.

Creating and submitting a job

Now that you have configured your analysis, you will create a job that is associated with your model and submit the job for analysis.

1. In the Model Tree, double-click the **Jobs** container to create a job.

   The **Create Job** dialog box appears.

2. Name the job `PullHinge`, and click **Continue**.

   The job editor appears.

3. In the **Description** field, type `Hinge tutorial`.

   Click the tabs to see the contents of the job editor, and review the default settings. Click **OK** to accept all the default job settings.

4. In the Model Tree, click mouse button 3 on the job named **PullHinge** and select **Submit** from the menu that appears to submit your job for analysis.

5. In the Model Tree, click mouse button 3 on the job name and select **Monitor** from the menu that appears to monitor the analysis as it runs.

   A warning may appear indicating that the surfaces associated with the hinge piece with a hole are no longer available. This is due to the partition created earlier. If this occurs, do one of the following before submitting the job:

   - If the contact pairs were defined manually, redefine the surfaces associated with the flange face and the flange hole.
   - If the contact pairs were defined automatically, simply delete the contact pairs and redefine the interactions as before.

   A dialog box appears with the name of your job in the title bar and a status chart for the analysis. Messages appear in the lower panel of the dialog box as the job progresses. Click the **Errors** and **Warnings** tabs to check for problems in the analysis.

   Once the analysis is underway, an *X*–*Y* plot of the values of the degree of freedom that you selected to monitor earlier in the tutorial appears in a separate window in the viewport. (You may need to resize the viewport windows to see it.) You can follow the progression of the node's displacement over time in the 1-direction as the analysis runs.

6. When the job completes successfully, the status of the job appearing in the Model Tree changes to `Completed`. You are now ready to view the results of the analysis with the Visualization module. In the Model Tree, click mouse button 3 on the job name and select **Results** from the menu that appears.

   Abaqus/CAE enters the Visualization module, opens the output database created by the job, and displays the undeformed shape of the model.

   Note:

   You can also enter the Visualization module by clicking **Visualization** in the **Module** list located in the context bar. However, in this case Abaqus/CAE requires you to open the output database explicitly using the **File** menu.

Viewing the results of your analysis

You will view the results of your analysis by drawing a contour plot of the deformed model.
You will then use display groups to display one of the hinge pieces; by displaying just a portion of the model you can view results that are not visible when you display the whole model.

In this section:

- *Displaying and customizing a contour plot*
- *Using display groups*

Displaying and customizing a contour plot

In this section you will display a contour plot of the model and adjust the deformation scale factor.

1. From the main menu bar, select **Plot->Contours->On Deformed Shape**.

   Abaqus/CAE displays a contour plot of von Mises stress superimposed on the deformed shape of the model at the end of the last increment of the loading step, as indicated by the following text in the state block:

   ```
   Step: Load, Apply load
   Increment     6: Step Time =   1.000
   ```

   By default, all surfaces with no results (in this case, the pin) are displayed in white.

   The deformation is exaggerated because of the default deformation scale factor that Abaqus/CAE selects.

2. To remove the white surfaces from the display, do the following:
   a. In the Results Tree, expand the **Surface Sets** container underneath the output database file named `PullHinge.odb`.
   b. Select all the surfaces that appear in the list.
   c. Click mouse button 3, and select **Remove** from the menu that appears.

      The white surfaces disappear from the view.

3. To reduce the deformation scale factor, do the following:
   a. From the main menu bar, select **Options->Common**.

      The **Common Plot Options** dialog box appears.

   b. From the **Deformation Scale Factor** options, choose **Uniform**.
   c. In the **Value** text field, type a value of `100`; and click **OK**.

      Abaqus/CAE displays the contour plot with a deformation scale factor of 100, as shown in *Figure 1*.

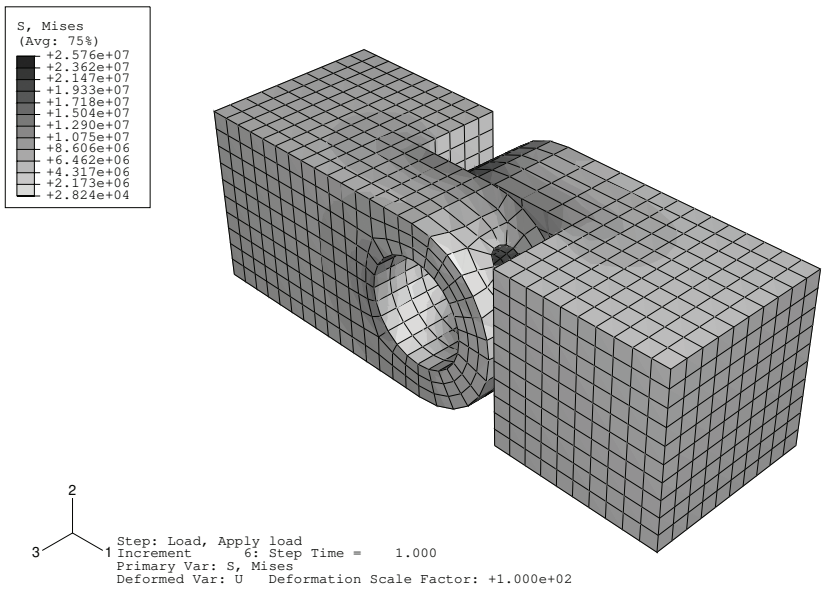


Figure 1: Contour plot of von Mises stress with a reduced deformation scale factor.

4. Use the view manipulation tools to examine the deformed model. Note where the pin appears to be exerting the most pressure against the insides of the flanges. Also note how the two flanges have twisted away from each other.
5. By default, the contour plot displays the von Mises stresses in the model; you can view other variables by selecting them from the **Field Output** toolbar. Select **S11** from the list of components and invariants on the right side of the **Field Output** toolbar.

   Abaqus/CAE replaces the default von Mises plot with a contour plot of the stresses in the 1-direction.
6. Select **Max. Principal** from the list of components and invariants to see the maximum principal stresses on the model.
7. Select any other variables of interest from the **Field Output** toolbar.
8. Click the  tool in the toolbar to display the **Field Output** dialog box. On the **Primary Variable** tab, select **S** as the output variable, select **Mises** as the invariant, and click **OK** to display the von Mises stresses again and to close the dialog box.

   The **Field Output** dialog box offers some controls and access to other dialog boxes, such as the **Section Points** dialog box, that cannot be accessed from the **Field Output** toolbar.

Using display groups

You will now create a display group that includes only the element sets that make up the hinge piece that includes the lubrication hole. By removing all other element sets from the display, you will be able to view results for the surface of the flange that contacts the other hinge.

1. In the Results Tree, expand the **Instances** container underneath the output database named `PullHinge.odb`.
2. From the list of available part instances, select **HINGE-HOLE-1**. Click mouse button 3, and select **Replace** from the menu that appears to replace the current display group with the selected elements.

   Click , if necessary, to fit the model in the viewport.

   The contour plot of the entire model is replaced by a plot of only the selected hinge piece, as shown in *Figure 1*.

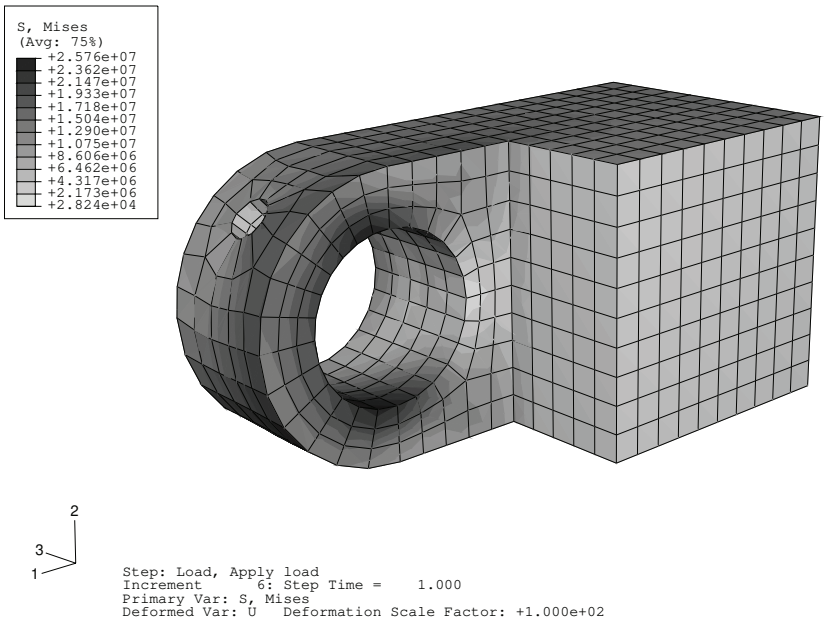


Figure 1: Use display groups to view a contour plot of the von Mises stress in the hinge piece with the lubrication hole.

3. Use the view manipulation tools to view the hinge at different angles. You can now see results for surfaces on the hinge that were hidden by the solid hinge.
4. From the main menu bar, select **Result->Field Output**.
5. From the top of the **Primary Variable** tabbed page, toggle on **List only variables with results:** and choose **at surface nodes** from the menu.
6. From the list of variables that appears, select **CPRESS**, and click **Apply**.

   Abaqus/CAE displays a contour plot of the contact pressures in the flange hole.

For more information about using the Visualization module, see the following sections:

- *Viewing the results of your analysis*
- *Viewing the Output from Your Analysis*

You have now completed this tutorial and learned how to:

- create and modify features;
- use datum geometry to add features to a model;
- use position constraints to assemble a model composed of more than one part;
- define contact interactions between regions of a model;
- monitor the progress of an analysis job; and
- use display groups to view results for individual parts of a model.

Summary

This section presents a summary of the concepts.

- When you create a part, you can create a deformable part, a discrete rigid surface, or an analytical rigid surface. You can subsequently change the type of the part.
- You can create parts by adding features to the base feature. When you add a feature, you must select a face on which to sketch the profile of the feature. When you delete a feature from a part, Abaqus/CAE also deletes any features that depend on the feature being deleted. These dependent features are called children.
- You can edit features by modifying the sketch of the feature or a parameter associated with the feature, such as an extrusion depth. Editing features can cause dependent features to fail during regeneration.
- The Datum toolset allows you to create datum points, axes, and planes. Datum geometry that you create on a part can also be used by the Sketcher. For example, if a suitable sketch plane does not exist, you can use the Datum toolset to create one.
- Click **OK** in a dialog box to perform the selected operation and to close the dialog box; click **Apply** to leave the dialog box open while performing the selected operation. Click **Cancel** to close the dialog box without performing an operation.
- You can use the tools in the View Manipulation toolbar to change the view of the model to a more convenient one. Use mouse button 2 to stop any view manipulation. If you rotate or pan the sketch, use the cycle view manipulation tool to restore the original view.
- You should save the model database at regular intervals.
- When you create a part instance, the default position is based on the sketch of the base feature. You can ask Abaqus/CAE to offset the new instance along the *X*-axis so that it does not overlap any existing instances. A graphic indicates the origin and the orientation of the global coordinate system in the Assembly module.
- You position part instances relative to each other in the Assembly module using a sequence of constraint operations.
- Part instances can be classified as either dependent or independent.
- You use the step editor to control the time incrementation during the step.
- You can use managers to display a list of the entities you have defined—for example, steps—and to help you perform repeated operations.
- By default, Abaqus/CAE propagates interactions or prescribed conditions defined in one step to all subsequent steps.
- Abaqus/CAE color codes the model to indicate how a region will be meshed. Green indicates that a region can be meshed with structured methods, yellow indicates that a region can be meshed with sweep methods, and orange indicates that a region cannot be meshed.
- You use the Partition toolset to divide the model into regions that Abaqus/CAE can mesh.
- When you create and name a job, Abaqus/CAE uses the same name for the input file it generates. Consequently, all files associated with the analysis (for example, the output database, the message file, and the status file) use the same name.
- You can view the progression of a degree of freedom over the course of an analysis that you have chosen to monitor before submitting the job.
- When you first open an output database, Abaqus/CAE displays an undeformed plot of the model.
- You use display groups to display selected regions of your model. A display group can be composed of any combination of selected part instances, geometry (cells, faces, or edges), elements, nodes, or surfaces.

Viewing the Output from Your Analysis

This postprocessing tutorial for the experienced Abaqus user illustrates how you can use the Visualization module (also licensed separately as Abaqus/Viewer) to display the results from your analysis in graphical form.

In this section:

- *Displaying output*
- *Which variables are in the output database?*
- *Reading the output database*
- *Customizing a model plot*
- *Displaying the deformed model shape*
- *Displaying and customizing a contour plot*
- *Animating a contour plot*
- *Displaying and customizing a symbol plot*
- *Displaying and customizing a material orientation plot*
- *Displaying and customizing an X–Y plot*
- *Operating on X–Y data*
- *Probing an X–Y plot*
- *Displaying results along a path*
- *Summary*

Displaying output

During the tutorial you will display the output from Case 2 of the example problem, *Indentation of an elastomeric foam specimen with a hemispherical punch*. The problem studies the behavior of a heavy metal punch impacting a soft elastomeric foam block; the resulting deformation and strain are shown in *Figure 1*.

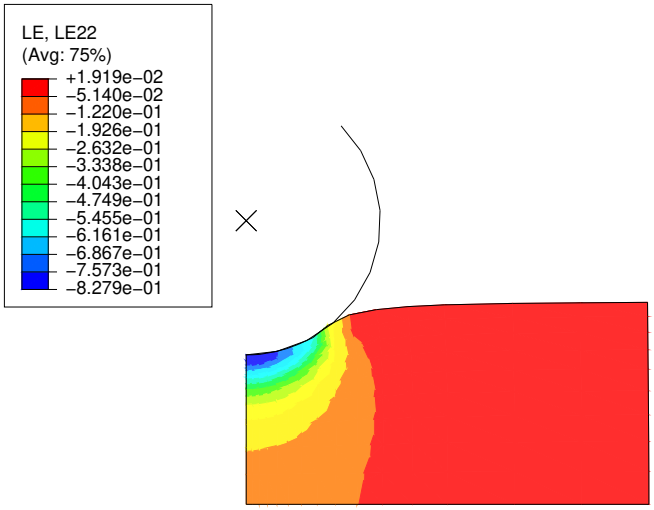


Figure 1: Contour plot showing deformation and strain.

The problem is modeled in two dimensions and is divided into three steps:

1. The punch initially rests on the surface of the foam block and compresses the block under its own weight. The gravity loading is ramped up over two seconds; but the analysis continues for a total of five seconds, allowing the foam to relax fully. The analysis uses the quasi-static procedure to model the response of the foam block during the step.
2. The punch is forced down with an impulsive load that varies according to a half sine wave over a period of one second. The response of the foam block is modeled using the implicit dynamic procedure.
3. The impulsive load is removed, and the punch is allowed to move freely while the foam expands and contracts. The viscoelastic foam damps out the vibrations, and the step runs for 10 seconds while the model returns to steady state. As with the second step, the response of the foam block is modeled using the implicit dynamic procedure.

The tutorial consists of the following sections:

- *Which variables are in the output database?*
- *Reading the output database*
- *Customizing a model plot*
- *Displaying the deformed model shape*
- *Displaying and customizing a contour plot*
- *Animating a contour plot*
- *Displaying and customizing a symbol plot*
- *Displaying and customizing a material orientation plot*
- *Displaying and customizing an X–Y plot*

- *Operating on X–Y data*
- *Probing an X–Y plot*
- *Displaying results along a path*

Which variables are in the output database?

In the first step of the elastomeric foam example, a set of options is included to control the data output during each step of the analysis. Abaqus/Standard writes this output to the Field Output or History Output portion of the output database, depending on the output type.

Field Output

The Field Output portion of the output database contains variables that should be output relatively infrequently during the analysis; in this case, after every 10 increments and after the last increment of a step. Typically, you select output for your entire model or a large region of your model, and Abaqus writes every component at the selected frequency. Only the selected variables are written to the output database.

The following input file fragment shows the options that control the field output variables in the elastomeric block example:

```
*OUTPUT, FIELD, FREQUENCY=10
*CONTACT OUTPUT, SECONDARY=ASURF, MAIN=BSURF,
VARIABLE=PRESELECT
*NODE OUTPUT
U,
*ELEMENT OUTPUT, ELSET=FOAM
S,E
```

Abaqus/Standard writes the following variables to the Field Output portion of the output database after every 10 increments and at the end of each step:

- the stress components of every integration point in the foam block;
- the logarithmic strain components of every integration point in the foam block (by default, the logarithmic strain is written to the output database when the user requests strain for a geometrically nonlinear analysis);
- the displacement of every node in the model; and
- the default contact output variables (clearance, pressure, shear stress, and tangential motion) resulting from the contact between the punch and the foam block.

History Output

The History Output portion of the output database contains variables that may be output relatively frequently during the analysis, as often as every increment. To avoid generating large amounts of data, you typically select output from a small area of your model, such as a single element or a small region. In addition, you must select the individual components of the variables that are written to the output database. History output is typically used for generating *X*–*Y* data plots.

The following input file fragment shows the options that control the history output variables in the elastomeric block example:

```
*OUTPUT, HISTORY, FREQUENCY=1
*NODE OUTPUT, NSET=N9999
U2, V2, A2
*ELEMENT OUTPUT, ELSET=CORNER
MISES, E22, S22
```

Abaqus/Standard writes the following variables from the punch's rigid body reference node (contained in node set N9999) to the history portion of the output database after every increment:

- the vertical displacement,

- the vertical velocity, and
- the vertical acceleration.

In addition, after every increment Abaqus/Standard writes the following variables from the element at the corner of the block to the history portion of the output database:

- von Mises stress,
- the logarithmic strain in the 2-direction on the 2-plane, and
- the stress in the 2-direction on the 2-plane.

The stress and strain variables are written for all the integration points in the element.

Reading the output database

To start the tutorial, open the output database that Abaqus/Standard generated during the analysis of the example problem.

1. You can use the Abaqus **fetch** utility to copy the output database to your local directory. Enter the following command at the operating system prompt:

   ```
   abaqus fetch job=viewer_tutorial
   ```

2. If you have not done so already, start Abaqus/CAE or Abaqus/Viewer by typing `abaqus cae` or `abaqus viewer`, respectively, at the operating system prompt.
3. From the **Start Session** dialog box that appears, select **Open Database**. If you are already in an Abaqus/CAE or Abaqus/Viewer session, select **File->Open** from the main menu bar.

   The **Open Database** dialog box appears.
4. From the **File Filter** list at the bottom of the **Open Database** dialog box, select **Output Database (*.odb)**.

   The remainder of the dialog box changes to reflect the fact that you are now interested in files with the extension `.odb` only.

   Note:

   If you are running Abaqus/Viewer, you do not have to select the file filter; only output database files are shown in the **Open Database** dialog box.

5. From the **Directory** list at the top of the **Open Database** dialog box, select your local directory.
6. From the list of output database files that appears, select `viewer_tutorial.odb`.
7. Click **OK**.

   Abaqus/CAE starts the Visualization module and displays the undeformed shape of the model, as shown in *Figure 1*.

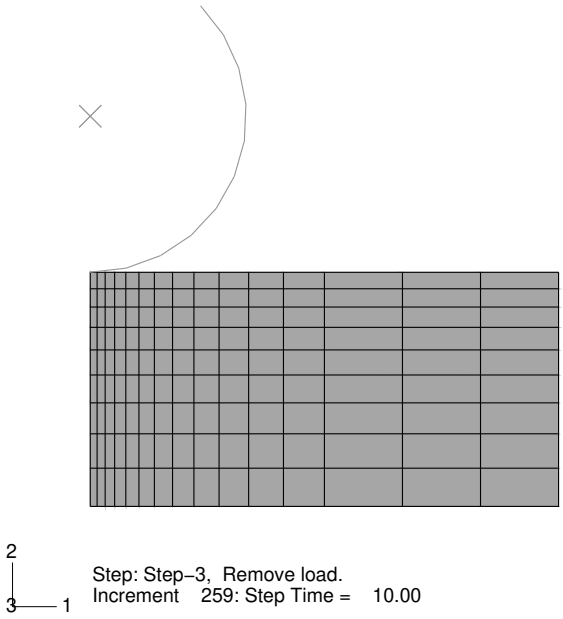


Figure 1: Undeformed model shape.

The title block at the bottom of the viewport indicates the following:

- The description of the model (from the first line of the **HEADING* option in the input file).
- The name of the output database (from the name of the analysis job).
- The product name (Abaqus/Standard or Abaqus/Explicit) used to generate the output database.
- The date the output database was last modified.

The state block at the bottom of the viewport indicates the following:

- Which step is being displayed.
- The increment within the step.
- The step time.

The view orientation triad indicates the orientation of the model in the global coordinate system. The 3D compass located in the upper-right corner of the viewport allows you to manipulate the view directly.

Note:

For clarity, most of the figures in this tutorial do not include the title block, state block, orientation triad, and 3D compass. When the triad does appear, however, the nondefault 1-2-3 triad labels are used. In general, the figures illustrate the effect on the model of changing and customizing the plot. You can toggle off and customize the title block, state block, and orientation triad by selecting **Viewport**->**Viewport Annotation Options** from the main menu bar.

Customizing a model plot

You will now use the plot options to request element numbering.

A common set of plot options for all plot states—undeformed, deformed, contour, etc.—is provided to allow you to customize the appearance of a plot. Regardless of the plot state, customization options apply only to the current viewport and are not saved between sessions.

1. From the main menu bar, select **Options->Common**.

   Abaqus displays the **Common Plot Options** dialog box.

2. By default, Abaqus fills the model in green and hides element labels. You will change the color of the element labels from cyan to red and display them. From the **Common Plot Options** dialog box, click the **Labels** tab and do the following:

   a. Toggle on **Show element labels**.

   b. Click **Apply**.

      Abaqus displays the element numbering using cyan text.

   c. Click the color sample  for the element labels.

      Abaqus displays the **Select Color** dialog box.

   d. Click the **RGB** tab and set the red, green, and blue values to 255, 0, and 0, respectively.

      Tip: You can also select red from the colors near the bottom of the dialog box or use any of the other available selection methods.

   e. Click **OK** to accept your selection.

      Abaqus closes the **Select Color** dialog box and updates the color sample to red.

   f. Click **OK**.

      The color of the element labels changes from cyan to red, and the **Common Plot Options** dialog box closes.

Displaying the deformed model shape

You can display a plot of your model showing the deformed shape during each frame of the analysis. When you request a deformed shape plot of data from a force-displacement analysis, Abaqus plots the nodal displacements by default; but you can display any nodal vector field output variable that is available on the output database. You can also use the plot options to customize the appearance of a deformed plot.

In this section:

- *Displaying a deformed shape plot*
- *Superimposing the undeformed shape on the deformed shape*

Displaying a deformed shape plot

Most procedures in Abaqus/Standard or Abaqus/Explicit write displacement to the output database by default and also select displacement for the nodal vector quantity to use as the default deformed variable. When Abaqus reads the output database, it uses the default deformed variable to determine the deformed model shape. In the elastomeric block example the user requested output of the displacements (`U`) for every node in the model after every 10 increments, and displacement was selected as the default deformed variable.

(Some procedures—for example, heat transfer—do not write nodal vector quantities to the output database by default and do not select a variable as the default deformed variable. Therefore, Abaqus cannot display a deformed plot, since in such cases the output database does not contain any variables that can be used to compute a deformed shape.)

1. From the main menu bar, select **Plot->Deformed Shape**.

   Tip: You can also plot the deformed model using the  tool in the Visualization module toolbox.

   Abaqus displays the deformed model in the same increment and step that it last displayed the undeformed model. The element labels also appear because the common plot options were used to display them.

2. Open the **Common Plot Options** dialog box. Click **Defaults** followed by **Apply** to restore and apply the default options.

   The state block indicates the default deformed variable being plotted (`U`) and the deformation scale factor (`1.000e+00`). Abaqus selects a default deformation scale factor of 1.0 for large-displacement analyses. (For small-displacement analyses Abaqus chooses the default scale factor to fit the viewport optimally.)

3. The buttons in the context bar allow you to move between frames of the analysis. In particular, the button on the far right of the context bar is the frame selector  tool; it allows you to drag a slider to choose the frame of interest.

   You can also move directly to a selected step and increment using the following technique:

   a. From the main menu bar, select **Result->Step/Frame**.

      Abaqus displays the **Step/Frame** dialog box.

   b. Select `Step 1`, `Increment 0`, and click **Apply**.

   c. The **Step/Frame** dialog box also displays the step time associated with an increment. Use the **Step/Frame** dialog box to display the deformed model approximately halfway through the second step.

4. Use a combination of the buttons in the context bar and the **Step/Frame** dialog box to view the deformed plot in different frames and in different steps.

5. Display the deformed model after the last increment of the third step (`Step 3` and `Step Time = 10.00`), as shown in *Figure 1*.

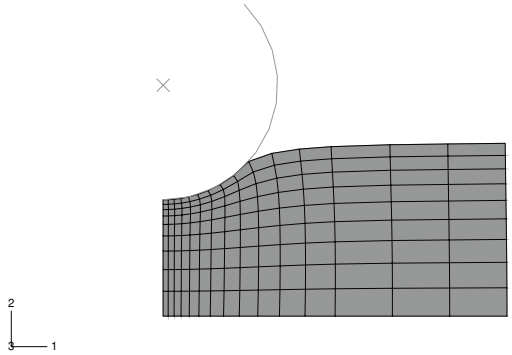


Figure 1: Deformed plot of the model after the last increment of the third step.

6. Click **Cancel** to close the **Step/Frame** dialog box.

Superimposing the undeformed shape on the deformed shape

You can plot the undeformed and deformed model shapes simultaneously.

1. Click the  tool in the toolbox to allow multiple plot states in the viewport. Then click the  tool or select **Plot->Undeformed Shape** to add the undeformed shape plot to the existing deformed plot in the viewport.

   Abaqus displays the deformed plot overlaid with the undeformed plot.

2. To customize the superimposed (i.e., undeformed) plot, select **Options->Superimpose** from the main menu bar.

   Abaqus displays the **Superimpose Plot Options** dialog box.

3. In the **Superimpose Plot Options** dialog box, select the **Other** tab.
4. In the **Other** tabbed page, select the **Translucency** tab. Toggle off **Apply Translucency**.
5. Next select the **Offset** tab. Select **Uniform**, and enter a value of `0.001` as the offset value.
6. Click **OK** to apply the changes and to close the **Superimpose Plot Options** dialog box.

   Abaqus displays the customized deformed plot, as shown in *Figure 1*. The undeformed model shape is colored white and is offset slightly from the deformed model shape to prevent the colors from overlapping.

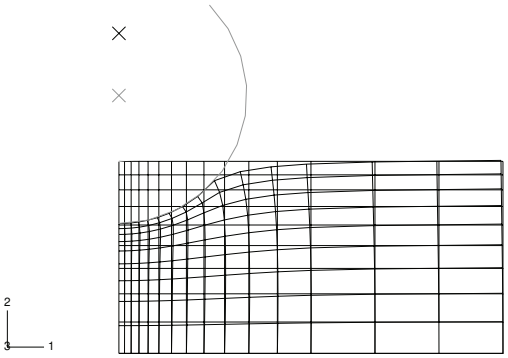


Figure 1: Customized deformed plot.

Displaying and customizing a contour plot

You can display a contour plot of your model showing a variable such as stress, strain, or temperature. In all plots, including contour plots, Abaqus selects a default variable to display. The default variable selected depends on the variables available in the output database, which in turn depend on the analysis procedures and the requested output. You can choose to display any variable that is available in the field output portion of the output database. If you select a variable when you are not in a plot state that can display that variable, a dialog box appears prompting you to switch to a valid plot state.

You can use the plot options to customize the appearance of a contour plot. Abaqus applies your customized settings to every contour plot displayed in the current viewport. If you display a contour plot in a new viewport, Abaqus reverts to the default plot options.

In this section:

- *Displaying a contour plot*
- *Selecting the variable to plot*
- *Customizing a contour plot*

Displaying a contour plot

You will first display a contour plot of the default variable. Before continuing, toggle off the multiple plot states option.

1. From the main menu bar, select **Plot->Contours->On Deformed Shape**.

    **Tip:** You can also display a contour plot using the  tool in the Visualization module toolbox.

   The state block indicates that the variable plotted is `S, MISES`, the default variable chosen by Abaqus. Abaqus displays the results at the same step and frame that you used to display the deformed shape plot.

2. Use the buttons in the context bar or the **Step/Frame** dialog box to view the contour plot in different frames and in different steps.

    **Note:**

   The legend changes as you move between frames. Abaqus updates the maximum and minimum values and computes the contour intervals in every frame.

Selecting the variable to plot

Abaqus selects a default variable to display in a contour plot, but you can display any variable that is available in the field output portion of the output database.

1. From the main menu bar, select **Result->Field Output**.

   Abaqus displays the **Field Output** dialog box. You use the **Field Output** dialog box to select the variable to display.

   Tip: You can also select most field output variables and their components or invariants by using the lists in the **Field Output** toolbar. For more information, see *Using the field output toolbar*.

2. Click the **Primary Variable** tab if it is not already selected.
3. To select the 22-component of strain as the primary variable, do the following:
   a. From the **Output Variable** field, select `LE` (logarithmic strain components at integration points).
   b. From the **Component** field, select the component `LE22`.
4. Click **OK** to select `LE22` as the primary variable and to close the **Field Output** dialog box.

   The contour plot in the current viewport changes to a plot of `LE22`, as shown in *Figure 1*.

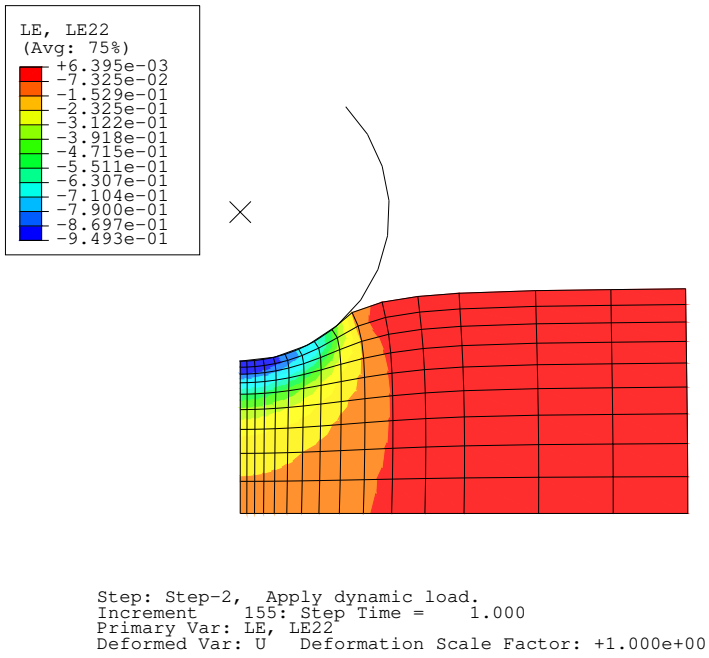


Figure 1: Contour plot of the model after the last increment of the third step.

Tip: You can click on the left side of the **Field Output** toolbar to make your selections from the **Field Output** dialog box instead of the toolbar. If you use the dialog box, you must click **Apply** or **OK** for Abaqus/CAE Abaqus/Viewer to display your selections in the viewport.

Customizing a contour plot

By default, Abaqus displays a contour plot using 12 equal intervals between the maximum and minimum value of the selected variable. Abaqus updates the maximum and minimum values and computes new contour intervals for every frame. The legend indicates the calculated intervals and the color corresponding to each interval. You can change the number of intervals, and you can set the values corresponding to the maximum and minimum contour limits. When you set the contour limits, Abaqus uses the values you supply in every contour plot displayed thereafter, regardless of the frame and which variable is being contoured.

1. Display the contour plot at the end of the last increment of the second step (`Step 2` and `Step Time = 1.000`).
2. From the main menu bar, select **Options->Contour**.

   Abaqus displays the **Contour Plot Options** dialog box.
3. Click the **Basic** tab if it is not already selected, and drag the uniform contour intervals slider to `16`.
4. Click the **Limits** tab to access the contour limits options.
   a. In the **Max** field, toggle the **Specify** button and type a maximum contour limit of `0.1`.
   b. In the **Min** field, toggle the **Specify** button and type a minimum of `-0.75`.
5. Click **Apply** to view the customized contour plot.

   The plot changes, as shown in *Figure 1*.

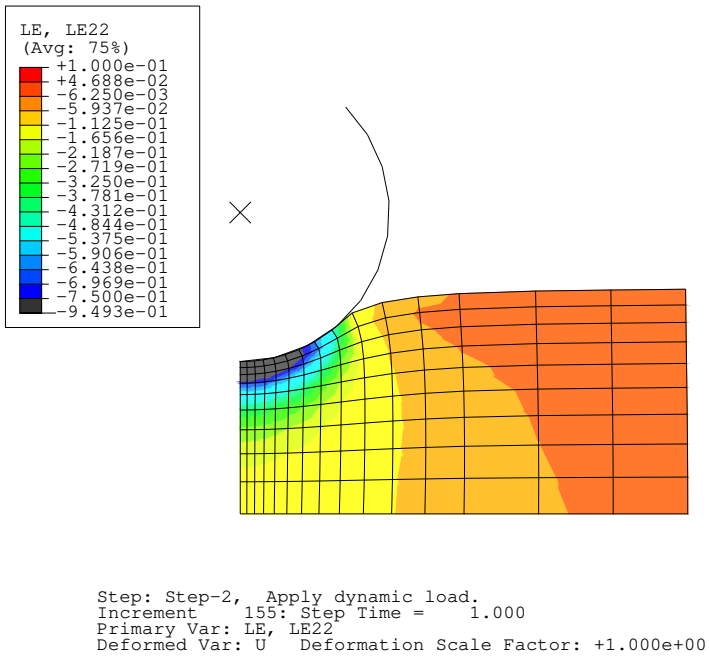


Figure 1: Customized contour plot.

Although you selected 16 contour intervals, the plot legend displays 17 intervals. Abaqus adds intervals to indicate any values that are greater than the maximum contour limit or less than the minimum contour limit and displays these values in light gray and dark gray, respectively. In this example, areas undergoing compressive strains greater than 0.75 are shown in dark gray. The minimum strain in the model is shown at the bottom of the contour legend. You might use either of these colors to indicate elements that fall outside the design range for the selected variable.

6. Under the **Limits** tab, examine the **Min** and **Max Auto-compute** options.

The minimum and maximum values of strain for the contour plot are shown next to the two **Auto-compute** options.

7. Toggle on **Show location** for the **Min/Max** options to display the locations in the model where the extreme values occur.
8. Click **OK** to close the **Contour Plot Options** dialog box.

Animating a contour plot

You can animate a deformed, contour, symbol, or material orientation (time history animation only) plot using one of the following:

Time History Animation

In a time history animation Abaqus displays each frame of each step from the output database in sequence, and you can see the change in the deformation or the change in a contour or symbol plot variable while the analysis progresses. In effect, Abaqus animates the results of the analysis. You can select which steps and frames to include in a time history animation.

Scale Factor Animation

Scale factor animation takes the results from a selected step and frame and simply scales them to form frames of the animation. You can select a scale factor that varies between zero and one or between minus one and plus one. Scale factor animation is particularly useful for animating vibration modes computed by an eigenvalue analysis.

The animation uses the plot options from the relevant mode—deformed, contour, symbol, or material orientation. In addition, you can control the following:

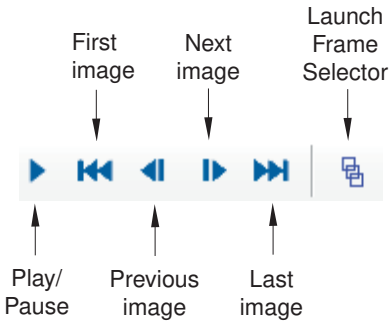
- The speed of the animation
- Whether the animation runs continuously or just once
- Whether to display the animation status

For the elastomeric foam example you will display a time history animation of a contour plot. The animated contour plot displays the field output variable displayed in the **Field Output** toolbar (`LE22`). In addition, it uses the same options that you selected for the contour plot; for example, the contour intervals and element edge display.

1. From the main menu bar, select **Animate->Time History**.

   Abaqus displays the customized contour plot at the beginning of the analysis and steps through each frame; the state block indicates the current step and increment throughout the animation. After the last increment of the last step, the animation restarts at the beginning of the analysis (`Step 1`, `Increment 0`, and `Step Time = 0.00`).

   Abaqus also displays the movie player controls on the right side of the context bar:

   You use these controls to play, pause, and step through the animation.

2. In the context bar, click the **Play/Pause** button to stop the animation.

The animation stops at the current image.

3. Click the button again to continue the animation.

   The animation resumes.

4. From the main menu bar, select **Options->Animation** to view the animation options.

   Abaqus displays the **Animation Options** dialog box.

5. Click the **Player** tab if it is not already selected, and do the following:

   a. Choose **Swing**.

   b. Click **OK**.

   Because you chose **Swing**, when the animation reaches the end of the analysis, it steps backward through each frame instead of jumping back to the beginning of the analysis.

6. You can also customize the contour plot while the animation is running.

   a. Display the **Contour Plot Options** dialog box.

   b. Reduce the number of contour intervals to 10.

   c. Click **OK** to apply your change and to close the **Contour Plot Options** dialog box.

7. When you have finished viewing the animation, click the **Play/Pause** button to stop the animation.

Displaying and customizing a symbol plot

Symbol plots allow you to visualize the magnitude and direction of vector and tensor variables in the form of symbols (arrows) superimposed on the model.
Each symbol starts at the location in the model where the value was obtained; symbols representing nodal quantities appear at nodes, and symbols representing integration point quantities appear at integration points. The length of the arrow indicates the magnitude of the vector or tensor, and the direction of the arrow indicates its direction.

For example, in this section you will create a symbol plot of displacement. The symbol plot displays arrows representing the magnitude and the direction of the displacement vector at each node.

In this section:

- *Displaying a vector symbol plot*
- *Customizing the symbol plot*

Displaying a vector symbol plot

Use the **Field Output** toolbar to specify the variable you want to plot. When you select a symbol variable from the toolbar, the Visualization module automatically switches the display to plot symbols on the deformed model shape.

1. From the list of variable types on the left side of the **Field Output** toolbar, select **Symbol**.
2. From the list of output variables in the center of the toolbar, select **U** (spatial displacement at nodes).
3. From the list of vector quantities and components on the right side of the toolbar, select **RESULTANT** as the vector quantity. This selection indicates that you want to plot the magnitudes of the displacement vectors.

   Abaqus displays the magnitude of the displacement vector on the deformed model shape on a symbol plot in the current viewport, as shown in *Figure 1*.



Tip: You can also display a symbol plot using the  tool in the Visualization module toolbox or by selecting **Plot->Symbols->On Deformed Shape**.

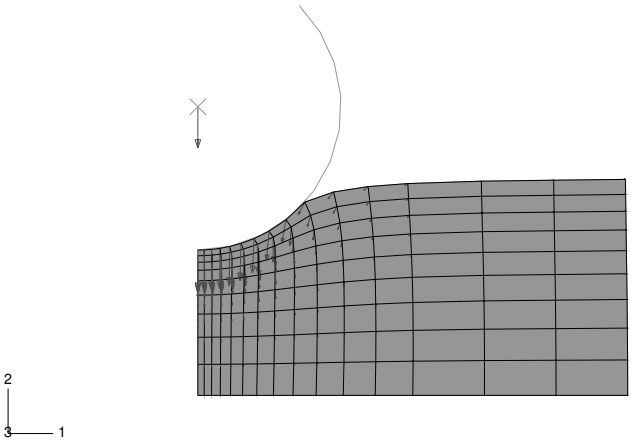


Figure 1: Symbol plot of displacement.

The arrows represent the total displacement at each node. The length of the arrow represents the magnitude of the displacement, and the direction of the arrow represents the direction of the displacement. The symbol plot legend shows how each arrow color corresponds to a specific range of values.

If your symbol plot is different from *Figure 1*, you may have selected the incorrect output variable.

Customizing the symbol plot

You will now customize your symbol plot by changing the visible edges and the arrow size and color.

1. From the main menu bar, select **Options->Common**.

   The **Common Plot Options** dialog box appears.

2. In the **Common Plot Options** dialog box, click the **Basic** tab if it is not already selected. Choose **Wireframe** for the render style and **Feature edges** for the visible edges.
3. From the main menu bar, select **Options->Symbol**.

   The **Symbol Plot Options** dialog box appears.

4. In the **Color & Style** tabbed page, do the following:
   a. Click the **Vector** tab.
   b. Set the color type to **Uniform**.
   c. Click the color sample .

      Abaqus/CAE displays the **Select Color** dialog box.
   d. Click the **RGB** tab and set the red, green, and blue values to 0, 255 and 255, respectively.

      Tip: You can also select cyan from the colors near the bottom of the dialog box or use any of the other available selection methods.

   e. Click **OK** to accept your selection and to close the **Select Color** dialog box.
   f. Drag the **Size** slider to select `12` as the maximum length of the vector.
5. Click **OK** to apply your changes and to close the **Symbol Plot Options** dialog box.

   The customized symbol plot appears, as shown in *Figure 1*.

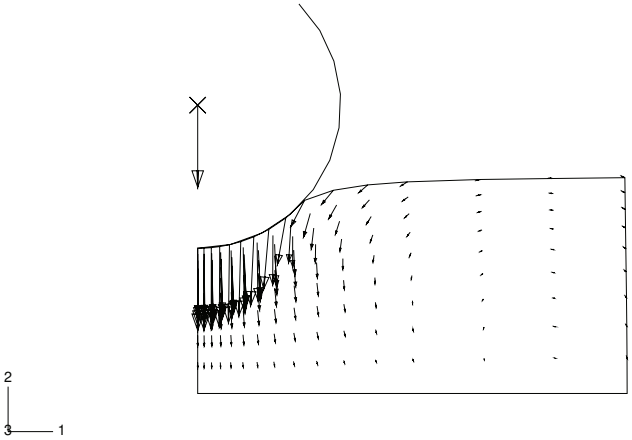


Figure 1: Customized symbol plot.

Displaying and customizing a material orientation plot

Material orientation plots allow you to visualize the material directions for each element in your model at a specified step and frame.
Material orientation triads that indicate the material directions are displayed at the element integration points.
Material orientation plots can be drawn on either the undeformed or the deformed shape of the model.

In this section you will create a material orientation plot on the deformed model shape and customize its appearance.

In this section:

- *Displaying a material orientation plot*
- *Customizing a material orientation plot*

Displaying a material orientation plot

The material orientation plot will be created at the step and frame of the analysis you specified previously.

From the main menu bar, select **Plot->Material Orientations->On Deformed Shape**.

Tip: You can also display a material orientation plot using the tool in the Visualization module toolbox.

A material orientation plot appears, as shown in *Figure 1*.

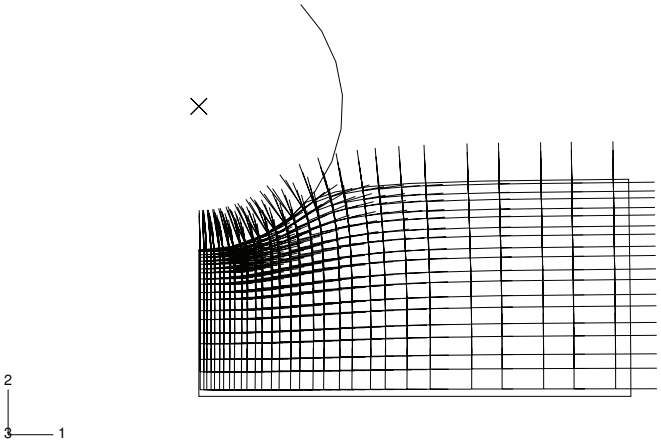


Figure 1: Material orientation plot.

Material orientation triads at element integration points indicate the material directions of each element in the model.

Customizing a material orientation plot

You will now customize your material orientation plot by changing the color and length of the material orientation triad axes.

1. From the main menu bar, select **Options->Material Orientation**.

   The **Material Orientation Plot Options** dialog box appears.

2. In this dialog box, do the following:

   a. Click the color sample for the 1-axis color.

   Abaqus/CAE displays the **Select Color** dialog box.

   b. Click the **RGB** tab; and set the red, green, and blue values to 255, 0, and 0, respectively.

   Tip: You can also select red from the colors near the bottom of the dialog box or use any of the other available selection methods.

   c. Click **OK** to accept your selection and to close the **Select Color** dialog box.

   d. Repeat the preceding three steps for the 2-axis color, changing it to blue (RGB 0, 0, 255).

   e. Drag the **Size** slider to reduce the length of the triad axes.

3. Click **OK** to apply your changes and to close the **Material Orientation Plot Options** dialog box.

   The customized material orientation plot appears, as shown in *Figure 1*.

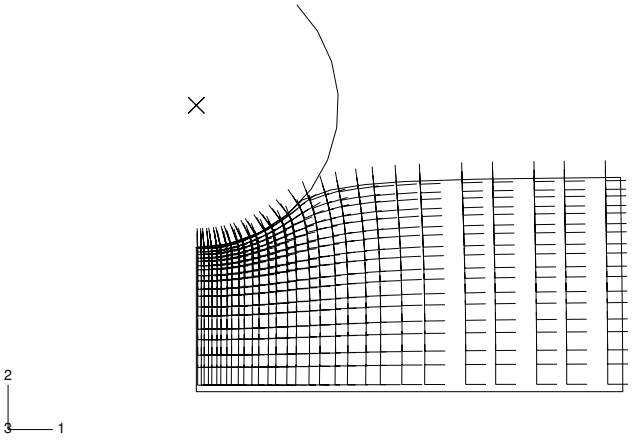


Figure 1: Customized material orientation plot.

Displaying and customizing an X–Y plot

You can display *X–Y* plots of data written to the output database.
For the tutorial you will display the vertical displacement of the rigid body reference node versus time.

The Visualization module also allows you to display *X–Y* plots of the following:

- Data read from an ASCII file.
- Data entered at the keyboard.
- Existing data, either combined with other data or arithmetically manipulated.

In this section:

- *Displaying an X–Y plot*
- *Customizing an X–Y plot*

Displaying an X–Y plot

You will now display an *X–Y* plot of displacement versus time.

1. In the Results Tree, click mouse button 3 on **History Output** for the output database named `viewer_tutorial.odb`. From the menu that appears, select **Filter**.
2. In the filter field, enter `*U2*` to restrict the history output to just the displacement components in the 2-direction.
3. Expand the **History Output** container and double-click the data object containing the history of the vertical motion of the rigid body reference node: `Spatial displacement: U2 at Node 1000 in NSET PUNCH`.

   Abaqus displays an *X–Y* plot of displacement versus time, as shown in *Figure 1*.

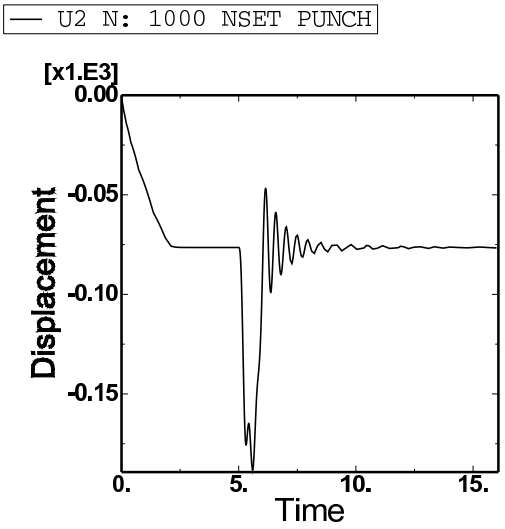


Figure 1: *X–Y* plot of displacement versus time.

   Default options selected by Abaqus include default ranges for the *X*- and *Y*-axes, axis titles, major and minor tick marks, the color of the line, and a legend.

4. The legend labels the *X–Y* plot **U2 N: 1000 NSET PUNCH**. This is a default name provided by Abaqus.

Customizing an X–Y plot

By default, Abaqus computes the range of the *X*- and *Y*-axes from the minimum and maximum values found in the data read from the output database. Abaqus divides each axis into intervals and displays the appropriate major and minor tick marks. The **Axis Options** allow you to set the range of each axis and to customize the appearance of the axes; the **Curve Options** allow you to customize the appearance of the individual curves; the **Chart Options** and **Chart Legend Options** allow you to position the grid and legend, respectively. *X*–*Y* plot customization options apply only to the current viewport and are not saved between sessions.

1. From the main menu bar, select **Options->XY Options->Axis** (or click  in the prompt area to cancel the current procedure, if necessary, and double-click either axis in the viewport).

   Abaqus displays the **Axis Options** dialog box.
2. Switch to the **Scale** tabbed page, if it is not already selected.
3. Specify that the *X*-axis should extend from `0` (the *X*-axis minimum) to `20` (the *X*-axis maximum) and that the *Y*-axis should extend from `-200` (the *Y*-axis minimum) to `0` (the *Y*-axis maximum).

    **Tip:** Select each axis in turn in the **Axis Options** dialog box, and then edit the scale as noted above.
4. From the options in the **Axis Options** dialog box, do the following.
   - In the **Scale** tabbed page, request that major tick marks appear on the *X*-axis at four-second increments (select **By increment** in the **Tick Mode** region of the page).
   - Request **3** minor tick marks per increment along the *X*-axis (this corresponds to a minor tick mark every second) and **4** minor tick marks per increment along the *Y*-axis (this corresponds to a minor tick mark every 10 mm).
   - In the **Title** tabbed page, type a *Y*-axis title of `Displacement U2 (mm)`.
   - In the **Axes** tabbed page, request a **Decimal** format with zero decimal places for the *Y*-axis labels.
5. Click **Dismiss** to close the **Axis Options** dialog box.
6. From the main menu bar, select **Options->XY Options->Chart** (or double-click any empty spot in the plot) to modify the gridlines and position the grid.
   a. In the **Chart Options** dialog box that appears, switch to the **Grid Display** tabbed page.
   b. Toggle on **Major** in both the **X Grid Lines** and **Y Grid Lines** fields. Change the color of the major gridlines to blue; the line style should be solid.
   c. Switch to the **Grid Area** tabbed page.
   d. In the **Size** region of this page, select the **Square** option.
   e. Use the slider to set the size to **75**.
   f. In the **Position** region of this page, select the **Auto-align** option.
   g. From the available alignment options, select the fourth to last one (position the grid in the bottom-center of the viewport).
   h. Click **Dismiss**.
7. From the main menu bar, select **Options->XY Options->Chart Legend** (or double-click the legend) to position the legend.
   a. In the **Chart Legend Options** dialog box, switch to the **Area** tabbed page.
   b. In the **Position** region of this page, toggle on **Inset** and click **Dismiss**.
   c. Drag the legend in the viewport to reposition it.

The customized *X–Y* plot appears, as shown in *Figure 1*.

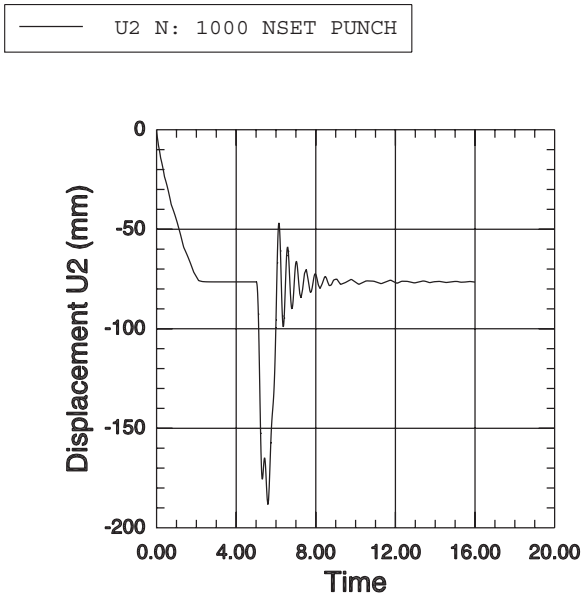


Figure 1: Customized *X–Y* plot of displacement.

8. You will now display a second *X–Y* plot in a new viewport. To create a new viewport, select **Viewport->Create** from the main menu bar.

   The new viewport appears. The same *X–Y* plot that you had in the first viewport appears in the new viewport.

   When multiple viewports are visible, the dark gray title bar indicates the current viewport; all work takes place in the current viewport. For more information, see *What is a viewport?*.

9. Tile the viewports vertically by selecting **Viewport->Tile Vertically** from the main menu bar.

10. Create a similar *X–Y* plot of vertical velocity (`V2`) versus time. You cannot select velocity during the first step because the first step was not a dynamic step; Abaqus/Standard computed velocity and acceleration only during the second and third steps. Use the same *X*-axis range as before, and use a *Y*-axis range from `1000` to `-1000`. Label the *Y*-axis `Velocity V2`. The finished plot is shown in *Figure 2*.

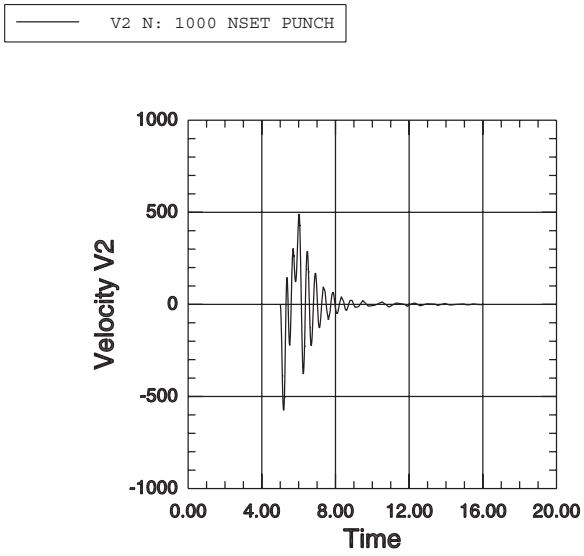


Figure 2: Customized *X–Y* plot of velocity.

Operating on X–Y data

An *X*–*Y* data object is a collection of ordered pairs that Abaqus stores in two columns—an *X*-column and a *Y*-column. The **Operate on XY Data** dialog box allows you to create new *X*–*Y* data objects by performing operations on previously saved *X*–*Y* data objects.
In this tutorial you will create a stress versus strain data object by combining a stress versus time data object with a strain versus time data object. Then, you will plot the stress-strain curve.

In this section:

- *Creating the stress versus time and strain versus time data objects*
- *Combining the data objects*
- *Plotting and customizing the stress-strain curve*

Creating the stress versus time and strain versus time data objects

The first step in creating the stress-strain curve is to create the stress versus time and the strain versus time data objects from the history output. The data objects will contain data from only the first step of the analysis, where the punch rests on the surface of the foam block and compresses the block under its own weight.

1. In the Results Tree, double-click the **XYData** container.

   The **Create XY Data** dialog box appears.

2. In this dialog box, select **ODB history output** if it is not already selected and click **Continue**.

   The **History Output** dialog box appears.

3. In the **History Output** dialog box, do the following:
   a. Click the **Variables** tab.
   b. Filter the data according to `*LE22*`.
   c. In the **Output Variables** field, select `Logarithmic strain components: LE22 at Element 1 Int Point 1 in ELSET CENT`.
   d. Click the **Steps/Frames** tab.
   e. Select `Step 1`.
   f. Click **Save As**.

      The **Save XY Data As** dialog box appears.

   g. Name the *X*–*Y* data `Strain`, and click **OK**.

   A data object called `Strain` containing logarithmic strain data (`LE22`) from integration point 1 of element 1 during the first step of the analysis appears in the **XYData** container.

4. Use a similar technique to create a data object containing stress data (`S22`) from integration point 1 of element 1 during the first step of the analysis. Name this data object `Stress`.

   Tip: Filter the variable list according to `*S22*`.

   Now you are ready to combine the two data objects to create a stress versus strain data object.

5. Dismiss the **History Output** dialog box.

Combining the data objects

In this section you will create a stress versus strain data object by combining the stress versus time and strain versus time data objects.

1. In the Results Tree, double-click the **XYData** container.
2. From the **Create XY Data** dialog box that appears, select **Operate on XY data** and click **Continue**.

   An **Operate on XY Data** dialog box appears. The dialog box contains the following lists:

   - The **XY Data** field on the left contains a list of existing *X*–*Y* data objects.
   - The **Operators** field on the right contains a list of all the possible operations you can perform on the data objects.

3. From the **Operators** field, click `combine(X,X)`.

   `combine( )` appears in the expression text field at the top of the dialog box.
4. In the **XY Data** field, drag the cursor across both the `Strain` and the `Stress` data objects to select both and click **Add to Expression** near the bottom of the dialog box.

   The expression `combine("Strain","Stress")` appears in the expression text field. In this expression `"Strain"` will determine the *X*-values and `"Stress"` will determine the *Y*-values in the combined plot.
5. From the buttons along the bottom of the **Operate on XY Data** dialog box, click **Save As**.
6. From the **Save XY Data As** dialog box that appears, enter the name `Stress-Strain` and click **OK**.

   The new data object `Stress-Strain` appears in the **XYData** container.
7. Cancel the **Operate on XY Data** dialog box.

Plotting and customizing the stress-strain curve

You will now use the `Stress-Strain` data object that you just created to plot the stress-strain curve.

1. In the **XYData** container, double-click `Stress-Strain`.

   A plot of the stress-strain curve appears in the viewport.

2. Your plot of stress versus strain inherited the customized chart settings from your previous plot. To restore the default chart options, do the following:

   a. Open the **Chart Options** dialog box.

   b. Toggle off **Major** in both the **X Grid Lines** and **Y Grid Lines** fields.

   c. Click **Dismiss**.

3. The plot of stress versus strain appears, as shown in *Figure 1*. In this figure, the visibility of the plot legend has been suppressed (open the **Chart Legend Options** dialog box; in the **Contents** tabbed page, toggle off **Show legend**).

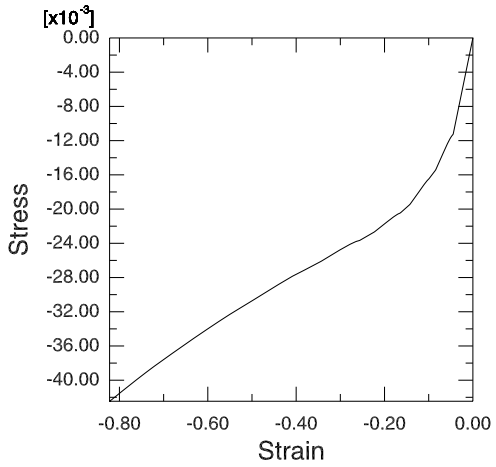


Figure 1: *X–Y* plot of stress versus strain.

Probing an X–Y plot

You can use the Query toolset in the Visualization module to probe your model and *X–Y* plots. You can also write the resulting probe values to a file. In this tutorial you will use the probe capability to obtain *X*- and *Y*-values from your stress–strain plot and to write these values to a file.

1. From the main menu bar, select **Tools**->**Query**; select **Probe values** from the **Query** dialog box.

   The **Probe Values** dialog box appears. Because an *X–Y* plot is in the current viewport, this dialog box will display *X–Y* curve data.

2. At the top of the dialog box, toggle on **Interpolate between points**. This option allows you to select arbitrary points along the curve.
3. In the viewport, position the cursor over the *X–Y* curve.

   When the arrow at the cursor approaches the *X–Y* curve, the point being probed is highlighted and information about it, including the corresponding *X–Y* coordinates, appears in the **Probe Values** table.

4. Click at various points along the curve.

   The points are added to the table in the **Probe Values** dialog box.

5. When you have finished selecting points, click **Write to File**.

   The **Report Probe Values** dialog box appears.

   By default, the data in the table are written to a file called `abaqus.rpt` in your current directory. The options in this dialog box allow you to change the name of this file and the format of the data written to the file.

6. Click **OK** to write your data to the file.
7. From the **Probe Values** dialog box, click **Cancel** to exit probe mode.

   A dialog box appears to inform you that the **Selected Probe Values** table contains data. Click **Yes** to indicate that it is OK to continue; the data in the table will be deleted.

Displaying results along a path

X–*Y* data can be generated for a specific path through your model.
In this tutorial you will specify a node list path along the top of the foam block and plot the displacement magnitude along this path.

In this section:

- *Creating a node list path*
- *Displaying results along a node list path*

Creating a node list path

A path is a line you define by specifying a series of points through the model. In a node list path all of the specified points are nodal locations. You create a node list path by entering node labels or node label ranges in a table. To determine the node labels of interest, it is helpful to create a model plot with the node labels visible.

1. Click the  tool to display the undeformed model shape.

   Use the **Common Plot Options** dialog box to display the node labels. Identify the nodes on the top edge of the foam block.

2. In the Results Tree, double-click **Paths**.

   The **Create Path** dialog box appears.

3. Name the path `Displacement`. Accept the default selection of **Node list** as the path type, and click **Continue**.

   The **Edit Node List Path** dialog box appears.

4. In the **Path Definition** table, select `PART-1-1` in the **Part Instance** field, type `1:601:40` in the **Node Labels** field, and press **[Enter]**. (This input specifies a range of nodes from 1 to 601 at increments of 40.) Alternatively, you can pick the nodes for the node list directly from the viewport by clicking **Add Before...**or **Add After...** in the **Edit Node List Path** dialog box.

   The selected path is highlighted in the plot in the current viewport.

5. Click **OK** to create the path and to close the **Edit Node List Path** dialog box.

Displaying results along a node list path

Abaqus obtains analysis results for each of the points on the path you have defined and generates *X*–*Y* data pairs; the *X*-values are the specified points in the model, and the *Y*-values are the analysis results at these points. You can generate an *X*–*Y* plot of the data pairs.

1. In the Results Tree, double-click the **XYData** container.
2. In the **Create XY Data** dialog box that appears, select **Path**; and click **Continue**.

   The **XY Data from Path** dialog box appears with the path that you created visible in the list of available paths.

   Accept the default selections in the **X Values** portion of the dialog box.
3. The result that will be plotted is displayed in the **Y Values** portion of the dialog box. If **U** is not indicated as the field output variable, click **Field Output** to change the variable.

   In the **Field Output** dialog box:

   a. Select **U** as the variable **Name**.
   b. Select **Magnitude** from the **Invariant** field.
   c. Click **OK**.
4. Click **Plot** to generate an *X*–*Y* plot of U along the path, as shown in *Figure 1*. You may need to reset the *X*–*Y* plot options to their default settings.

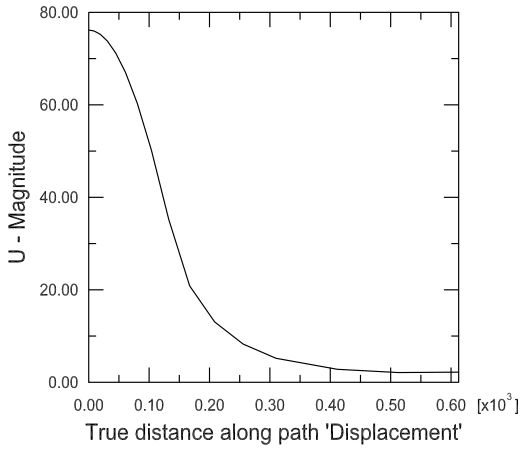


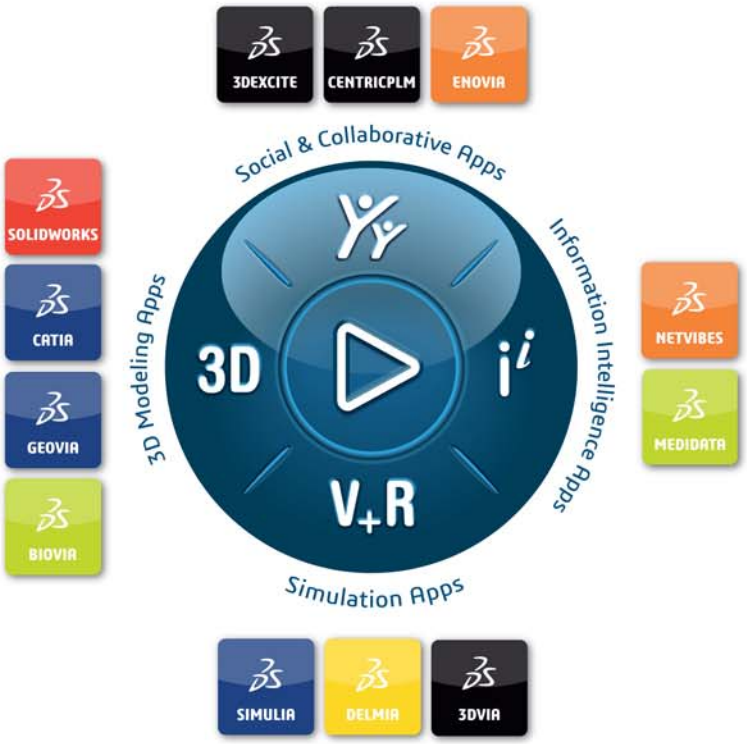
Figure 1: Path plot of U along the top of the foam block.

You have now finished the tutorial.

Summary

This section presents a summary of the concepts.

- Abaqus/CAE loads the Visualization module automatically when you open an output database.
- To perform many Visualization module functions, you can use either a menu item or a tool in the toolbox.
- You can use the buttons on the right side of the context bar to display the state of the model in each frame of the analysis.
- The Visualization module has different plot types. Common plot options are available to control the appearance of the model in all plot types. Some plot types have specialized plot options as well. Customization options apply only to the current viewport and are not saved between sessions. You can use the **Defaults** button to restore the default plot options.
- You use the viewport annotation options to customize the appearance of items that appear in all plots, such as the title block, the state block, and the orientation triad. The title block displays information about the analysis that generated the output database. The state block contains information about the step and increment being displayed.
- In all plots Abaqus selects a default variable to display from the field output portion of the output database. You can use the **Field Output** toolbar or dialog box to select the variable to display.
- You can display a time history animation from the data in an output database, or you can generate a scale factor animation based on a single increment of the results. You can animate a deformed, contour, symbol, or material orientation (time history animation only) plot; the animation uses the respective plot options to control the appearance of the model. You can customize these plots while the animation is running.
- A symbol plot shows the magnitude and direction of a particular vector or tensor variable at a specified step and frame. By default, symbol plots display the magnitudes for vector variables and all principal components for tensor variables.
- A material orientation plot shows the material directions of elements in your model at a specified step and frame of your analysis. Material orientations are displayed on an element-by-element basis at the material integration points, with no averaging across elements.
- You can display an *X*–*Y* plot of any variable stored in the output database. In most cases the *X*-axis is assumed to be time.
- You can use the **Operate on XY Data** dialog box to create new *X*–*Y* data objects based on operations on existing data objects.
- You can use the Query toolset to probe a model or *X*–*Y* plot. You can write the values you obtain to a file.



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 **3DEXPERIENCE** 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

©2025 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.